	1100010100_1000111010	// W0668_2048 = -0.460539	-0.887640
	1100010011_1000111010	// W0669_2048 = -0.463260	-0.886223
	1100010001_1000111011	// W0670_2048 = -0.465976	-0.884797
	1100001111_1000111100	// W0672_2048 = -0.471397	-0.881921
5	1100001100_1000111110	// W0674_2048 = -0.476799	-0.879012
	1100001010_1000111111	// W0675_2048 = -0.479494	-0.877545
	1100001001_1000111111	// W0676_2048 = -0.482184	-0.876070
	1100000110_1001000001	// W0678_2048 = -0.487550	-0.873095
	1100000100_1001000011	// W0680_2048 = -0.492898	-0.870087
10	1100000010_1001000011	// W0681_2048 = -0.495565	-0.868571
	1100000001_1001000100	// W0682_2048 = -0.498228	-0.867046
	1011111110_1001000110	// W0684_2048 = -0.503538	-0.863973
	1011111011_1001000111	// W0686_2048 = -0.508830	-0.860867
	1011111010_1001001000	// W0687_2048 = -0.511469	-0.859302
15	1011111001_1001001001	// W0688_2048 = -0.514103	-0.857729
	1011110110_1001001010	// W0690_2048 = -0.519356	-0.854558
	1011110011_1001001100	// W0692_2048 = -0.524590	-0.851355
	1011110010_1001001101	// W0693_2048 = -0.527199	-0.849742
	1011110001_1001001110	// W0694_2048 = -0.529804	-0.848120
20	1011101110_1001001111	// W0696_2048 = -0.534998	-0.844854
	1011101011_1001010001	// W0698_2048 = -0.540171	-0.841555
	1011101010_1001010010	// W0699_2048 = -0.542751	-0.839894
	1011101001_1001010011	// W0700_2048 = -0.545325	-0.838225
	1011100110_1001010101	// W0702_2048 = -0.550458	-0.834863
25	1011100100_1001010110	// W0704_2048 = -0.555570	-0.831470
	1011100010_1001010111	// W0705_2048 = -0.558119	-0.829761
	1011100001_1001011000	// W0706_2048 = -0.560662	-0.828045
	1011011110_1001011010	// W0708_2048 = -0.565732	-0.824589
	1011011100_1001011100	// W0710_2048 = -0.570781	-0.821103
30	1011011010_1001011100	// W0711_2048 = -0.573297	-0.819348
	1011011001_1001011101	// W0712_2048 = -0.575808	-0.817585
	1011010111_1001011111	// W0714_2048 = -0.580814	-0.814036
	1011010100_1001100001	// W0716_2048 = -0.585798	-0.810457
	1011010011_1001100010	// W0717_2048 = -0.588282	-0.808656
35	1011010010_1001100011	// W0718_2048 = -0.590760	-0.806848
	1011001111_1001100101	// W0720_2048 = -0.595699	-0.803208
	1011001100_1001100111	// W0722_2048 = -0.600616	-0.799537
	1011001011_1001101000	// W0723_2048 = -0.603067	-0.797691
	1011001010_1001101001	// W0724_2048 = -0.605511	-0.795837
40	1011000111_1001101010	// W0726_2048 = -0.610383	-0.792107
	1011000101_1001101100	// W0728_2048 = -0.615232	-0.788346
	1011000100_1001101101	// W0729_2048 = -0.617647	-0.786455
	1011000011_1001101110	// W0730_2048 = -0.620057	-0.784557
	1011000000_1001110000	// W0732_2048 = -0.624859	-0.780737
45	1010111110_1001110010	// W0734_2048 = -0.629638	-0.776888
	1010111100_1001110011	// W0735_2048 = -0.632019	-0.774953
	1010111011_1001110100	// W0736_2048 = -0.634393	-0.773010
	1010111001_1001110110	// W0738_2048 = -0.639124	-0.769103
	1010110110_1001111000	// W0740_2048 = -0.643832	-0.765167
50	1010110101_1001111001	// W0741_2048 = -0.646176	-0.763188
	1010110100_1001111010	// W0742_2048 = -0.648514	-0.761202
	1010110010_1001111100	// W0744_2048 = -0.653173	-0.757209
	1010101111_1001111110	// W0746_2048 = -0.657807	-0.753187
	1010101110_1001111111	// W0747_2048 = -0.660114	-0.751165
55	1010101101_1010000000	// W0748_2048 = -0.662416	-0.749136
	1010101010_1010000011	// W0750_2048 = -0.667000	-0.745058

```
       1110010000_1000001100   // W0584_2048 = -0.219101    -0.975702
       1110001110_1000001101   // W0585_2048 = -0.222094    -0.975025
       1110001101_1000001101   // W0586_2048 = -0.225084    -0.974339
       1110001010_1000001110   // W0588_2048 = -0.231058    -0.972940
  5    1110000111_1000001111   // W0590_2048 = -0.237024    -0.971504
       1110000101_1000001111   // W0591_2048 = -0.240003    -0.970772
       1110000100_1000001111   // W0592_2048 = -0.242980    -0.970031
       1110000001_1000010000   // W0594_2048 = -0.248928    -0.968522
       1101111110_1000010001   // W0596_2048 = -0.254866    -0.966976
 10    1101111100_1000010001   // W0597_2048 = -0.257831    -0.966190
       1101111010_1000010010   // W0598_2048 = -0.260794    -0.965394
       1101110111_1000010011   // W0600_2048 = -0.266713    -0.963776
       1101110100_1000010011   // W0602_2048 = -0.272621    -0.962121
       1101110011_1000010100   // W0603_2048 = -0.275572    -0.961280
 15    1101110001_1000010100   // W0604_2048 = -0.278520    -0.960431
       1101101110_1000010101   // W0606_2048 = -0.284408    -0.958703
       1101101011_1000010110   // W0608_2048 = -0.290285    -0.956940
       1101101010_1000010111   // W0609_2048 = -0.293219    -0.956045
       1101101000_1000010111   // W0610_2048 = -0.296151    -0.955141
 20    1101100101_1000011000   // W0612_2048 = -0.302006    -0.953306
       1101100010_1000011001   // W0614_2048 = -0.307850    -0.951435
       1101100001_1000011001   // W0615_2048 = -0.310767    -0.950486
       1101011111_1000011010   // W0616_2048 = -0.313682    -0.949528
       1101011100_1000011011   // W0618_2048 = -0.319502    -0.947586
 25    1101011001_1000011100   // W0620_2048 = -0.325310    -0.945607
       1101011000_1000011100   // W0621_2048 = -0.328210    -0.944605
       1101010110_1000011101   // W0622_2048 = -0.331106    -0.943593
       1101010100_1000011110   // W0624_2048 = -0.336890    -0.941544
       1101010001_1000011111   // W0626_2048 = -0.342661    -0.939459
 30    1101001111_1000100000   // W0627_2048 = -0.345541    -0.938404
       1101001110_1000100000   // W0628_2048 = -0.348419    -0.937339
       1101001011_1000100001   // W0630_2048 = -0.354164    -0.935184
       1101001000_1000100010   // W0632_2048 = -0.359895    -0.932993
       1101000110_1000100011   // W0633_2048 = -0.362756    -0.931884
 35    1101000101_1000100011   // W0634_2048 = -0.365613    -0.930767
       1101000010_1000100101   // W0636_2048 = -0.371317    -0.928506
       1100111111_1000100110   // W0638_2048 = -0.377007    -0.926210
       1100111110_1000100110   // W0639_2048 = -0.379847    -0.925049
       1100111100_1000100111   // W0640_2048 = -0.382683    -0.923880
 40    1100111001_1000101000   // W0642_2048 = -0.388345    -0.921514
       1100110110_1000101001   // W0644_2048 = -0.393992    -0.919114
       1100110101_1000101010   // W0645_2048 = -0.396810    -0.917901
       1100110011_1000101011   // W0646_2048 = -0.399624    -0.916679
       1100110001_1000101100   // W0648_2048 = -0.405241    -0.914210
 45    1100101110_1000101101   // W0650_2048 = -0.410843    -0.911706
       1100101100_1000101110   // W0651_2048 = -0.413638    -0.910441
       1100101011_1000101111   // W0652_2048 = -0.416430    -0.909168
       1100101000_1000110000   // W0654_2048 = -0.422000    -0.906596
       1100100101_1000110001   // W0656_2048 = -0.427555    -0.903989
 50    1100100100_1000110010   // W0657_2048 = -0.430326    -0.902673
       1100100010_1000110011   // W0658_2048 = -0.433094    -0.901349
       1100011111_1000110100   // W0660_2048 = -0.438616    -0.898674
       1100011101_1000110101   // W0662_2048 = -0.444122    -0.895966
       1100011011_1000110110   // W0663_2048 = -0.446869    -0.894599
 55    1100011010_1000110111   // W0664_2048 = -0.449611    -0.893224
       1100010111_1000111000   // W0666_2048 = -0.455084    -0.890449
```

```
      0000001101_10000. _ 00   // W0504_2048 = +0.024541    -0.999699
      0000001011_100000 :000  ·// W0505_2048 = +0.021474    -0.999769
      0000001001_1000000000    // W0506_2048 = +0.018407    -0.999831
      0000001000_1000000000    // W0507_2048 = +0.015339    -0.999882
  5   0000000110_1000000000    // W0508_2048 = +0.012272    -0.999925
      0000000101_1000000000    // W0509_2048 = +0.009204    -0.999958
      0000000011_1000000000    // W0510_2048 = +0.006136    -0.999981
      0000000010_1000000000    // W0511_2048 = +0.003068    -0.999995
      0000000000_1000000000    // W0512_2048 = +0.000000    -1.000000
 10   1111111110_1000000000    // W0513_2048 = -0.003068    -0.999995
      1111111101_1000000000    // W0514_2048 = -0.006136    -0.999981
      1111111010_1000000000    // W0516_2048 = -0.012272    -0.999925
      1111110111_1000000000    // WC0518_2048 = -0.013407   -0.999831
      1111110101_1000000000    // W0519_2048 = -0.021474    -0.999769
 15   1111110011_1000000000    // W0520_2048 = -0.024541    -0.999699
      1111110000_1000000000    // W0522_2048 = -0.030675    -0.999529
      1111101101_1000000000    // W0524_2048 = -0.036807    -0.999322
      1111101100_1000000000 ·  // W0525_2048 = -0.039873    -0.999205
      1111101010_1000000000    // W0526_2048 = -0.042938    -0.999078
 20   1111100111_1000000001    // W0528_2048 = -0.049068    -0.998795
      1111100100_1000000001    // W0530_2048 = -0.055195    -0.998476
      1111100010_1000000001    // W0531_2048 = -0.058258    -0.998302
      1111100001_1000000001    // W0532_2048 = -0.061321    -0.998118
      1111011101_1000000001    // W0534_2048 = -0.067444    -0.997723
 25   1111011010_1000000001    // W0536_2048 = -0.073565    -0.997290
      1111011001_1000000010    // W0537_2048 = -0.076624    -0.997060
      1111010111_1000000010    // W0538_2048 = -0.079682    -0.996820
      1111010100_1000000010    // W0540_2048 = -0.085797    -0.996313
      1111010001_1000000010    // W0542_2048 = -0.091909    -0.995767
 30   1111001111_1000000010    // W0543_2048 = -0.094963    -0.995481
      1111001110_1000000010    // W0544_2048 = -0.098017    -0.995185
      1111001011_1000000011    // W0546_2048 = -0.104122    -0.994565
      1111001000_1000000011    // W0548_2048 = -0.110222    -0.993907
      1111000110_1000000011    // W0549_2048 = -0.113271    -0.993564
 35   1111000100_1000000011    // W0550_2048 = -0.116319    -0.993212
      1111000001_1000000100    // W0552_2048 = -0.122411    -0.992480
      1110111110_1000000100    // W0554_2048 = -0.128498    -0.991710
      1110111101_1000000100    // W0555_2048 = -0.131540    -0.991311
      1110111011_1000000101    // W0556_2048 = -0.134581    -0.990903
 40   1110111000_1000000101    // W0558_2048 = -0.140658    -0.990058
      1110110101_1000000110    // W0560_2048 = -0.146730    -0.989177
      1110110011_1000000110    // W0561_2048 = -0.149765    -0.988722
      1110110010_1000000110    // W0562_2048 = -0.152797    -0.988258
      1110101111_1000000111    // W0564_2048 = -0.158858    -0.987301
 45   1110101100_1000000111    // W0566_2048 = -0.164913    -0.986308
      1110101010_1000000111    // W0567_2048 = -0.167938    -0.985798
      1110101000_1000001000    // W0568_2048 = -0.170962    -0.985278
      1110100101_1000001000    // W0570_2048 = -0.177004    -0.984210
      1110100010_1000001001    // W0572_2048 = -0.183040    -0.983105
 50   1110100001_1000001001    // W0573_2048 = -0.186055    -0.982539
      1110011111_1000001001    // W0574_2048 = -0.189069    -0.981964
      1110011100_1000001010    // W0576_2048 = -0.195090    -0.980785
      1110011001_1000001010    // W0578_2048 = -0.201105    -0.979570
      1110010111_1000001011    // W0579_2048 = -0.204109    -0.978948
 55   1110010110_1000001011    // W0580_2048 = -0.207111    -0.978317
      1110010011_1000001100    // W0582_2048 = -0.213110    -0.977028
```

```
     0001100100_1000001010  // W0448_2048 = +0.195090      -0.980785
     0001100010_1000001010  // W0449_2048 = +0.192080      -0.981379
     0001100001_1000001001  // W0450_2048 = +0.189069      -0.981964
     0001011111_1000001001  // W0451_2048 = +0.186055      -0.982539
5    0001011110_1000001001  // W0452_2048 = +0.183040      -0.983105
     0001011100_1000001000  // W0453_2048 = +0.180023      -0.983662
     0001011011_1000001000  // W0454_2048 = +0.177004      -0.984210
     0001011001_1000001000  // W0455_2048 = +0.173984      -0.984749
     0001011000_1000001000  // W0456_2048 = +0.170962      -0.985278
10   0001010110_1000000111  // W0457_2048 = +0.167938      -0.985798
     0001010100_1000000111  // W0458_2048 = +0.164913      -0.986308
     0001010011_1000000111  // W0459_2048 = +0.161886      -0.986809
     0001010001_1000000111  // W0460_2048 = +0.158858      -0.987301
     0001010000_1000000110  // W0461_2048 = +0.155828      -0.987784
15   0001001110_1000000110  // W0462_2048 = +0.152797      -0.988258
     0001001101_1000000110  // W0463_2048 = +0.149765      -0.988722
     0001001011_1000000110  // W0464_2048 = +0.146730      -0.989177
     0001001010_1000000101  // W0465_2048 = +0.143695      -0.989622
     0001001000_1000000101  // W0466_2048 = +0.140658      -0.990058
20   0001000110_1000000101  // W0467_2048 = +0.137620      -0.990485
     0001000101_1000000101  // W0468_2048 = +0.134581      -0.990903
     0001000011_1000000100  // W0469_2048 = +0.131540      -0.991311
     0001000010_1000000100  // W0470_2048 = +0.128498      -0.991710
     0001000000_1000000100  // W0471_2048 = +0.125455      -0.992099
25   0000111111_1000000100  // W0472_2048 = +0.122411      -0.992480
     0000111101_1000000100  // W0473_2048 = +0.119365      -0.992850
     0000111100_1000000011  // W0474_2048 = +0.116319      -0.993212
     0000111010_1000000011  // W0475_2048 = +0.113271      -0.993564
     0000111000_1000000011  // W0476_2048 = +0.110222      -0.993907
30   0000110111_1000000011  // W0477_2048 = +0.107172      -0.994240
     0000110101_1000000011  // W0478_2048 = +0.104122      -0.994565
     0000110100_1000000011  // W0479_2048 = +0.101070      -0.994879
     0000110010_1000000010  // W0480_2048 = +0.098017      -0.995185
     0000110001_1000000010  // W0481_2048 = +0.094963      -0.995481
35   0000101111_1000000010  // W0482_2048 = +0.091909      -0.995767
     0000101101_1000000010  // W0483_2048 = +0.088854      -0.996045
     0000101100_1000000010  // W0484_2048 = +0.085797      -0.996313
     0000101010_1000000010  // W0485_2048 = +0.082740      -0.996571
     0000101001_1000000010  // W0486_2048 = +0.079682      -0.996820
40   0000100111_1000000010  // W0487_2048 = +0.076624      -0.997060
     0000100110_1000000001  // W0488_2048 = +0.073565      -0.997290
     0000100100_1000000001  // W0489_2048 = +0.070505      -0.997511
     0000100011_1000000001  // W0490_2048 = +0.067444      -0.997723
     0000100001_1000000001  // W0491_2048 = +0.064383      -0.997925
45   0000011111_1000000001  // W0492_2048 = +0.061321      -0.998118
     0000011110_1000000001  // W0493_2048 = +0.058258      -0.998302
     0000011100_1000000001  // W0494_2048 = +0.055195      -0.998476
     0000011011_1000000001  // W0495_2048 = +0.052132      -0.998640
     0000011001_1000000001  // W0496_2048 = +0.049068      -0.998795
50   0000011000_1000000001  // W0497_2048 = +0.046003      -0.998941
     0000010110_1000000000  // W0498_2048 = +0.042938      -0.999078
     0000010100_1000000000  // W0499_2048 = +0.039873      -0.999205
     0000010011_1000000000  // W0500_2048 = +0.036807      -0.999322
     0000010001_1000000000  // W0501_2048 = +0.033741      -0.999431
55   0000010000_1000000000  // W0502_2048 = +0.030675      -0.999529
     0000001110_1000000000  // W0503_2048 = +0.027608      -0.999619
```

```
      0010111000_1000100010    // W0392_2048 = +0.359895    -0.932993
      0010110111_1000100010    // W0393_2048 = +0.357031    -0.934093
      0010110101_1000100001    // W0394_2048 = +0.354164    -0.935184
      0010110100_1000100001    // W0395_2048 = +0.351293    -0.936266
   5  0010110010_1000100000    // W0396_2048 = +0.348419    -0.937339
      0010110001_1000100000    // W0397_2048 = +0.345541    -0.938404
      0010101111_1000011111    // W0398_2048 = +0.342661    -0.939459
      0010101110_1000011110    // W0399_2048 = +0.339777    -0.940506
      0010101100_1000011110    // W0400_2048 = +0.336890    -0.941544
  10  0010101011_1000011101    // W0401_2048 = +0.334000    -0.942573
      0010101010_1000011101    // W0402_2048 = +0.331106    -0.943593
      0010101000_1000011100    // W0403_2048 = +0.328210    -0.944605
      0010100111_1000011100    // W0404_2048 = +0.325310    -0.945607
      0010100101_1000011011    // W0405_2048 = +0.322408    -0.946601
  15  0010100100_1000011011    // W0406_2048 = +0.319502    -0.947586
      0010100010_1000011010    // W0407_2048 = +0.316593    -0.948561
      0010100001_1000011010    // W0408_2048 = +0.313682    -0.949528
      0010011111_1000011001    // W0409_2048 = +0.310767    -0.950486
      0010011110_1000011001    // W0410_2048 = +0.307850    -0.951435
  20  0010011100_1000011000    // W0411_2048 = +0.304929    -0.952375
      0010011011_1000011000    // W0412_2048 = +0.302006    -0.953306
      0010011001_1000010111    // W0413_2048 = +0.299080    -0.954228
      0010011000_1000010111    // W0414_2048 = +0.296151    -0.955141
      0010010110_1000010111    // W0415_2048 = +0.293219    -0.956045
  25  0010010101_1000010110    // W0416_2048 = +0.290285    -0.956940
      0010010011_1000010110    // W0417_2048 = +0.287347    -0.957826
      0010010010_1000010101    // W0418_2048 = +0.284408    -0.958703
      0010010000_1000010101    // W0419_2048 = +0.281465    -0.959572
      0010001111_1000010100    // W0420_2048 = +0.278520    -0.960431
  30  0010001101_1000010100    // W0421_2048 = +0.275572    -0.961280
      0010001100_1000010011    // W0422_2048 = +0.272621    -0.962121
      0010001010_1000010011    // W0423_2048 = +0.269668    -0.962953
      0010001001_1000010011    // W0424_2048 = +0.266713    -0.963776
      0010000111_1000010010    // W0425_2048 = +0.263755    -0.964590
  35  0010000110_1000010010    // W0426_2048 = +0.260794    -0.965394
      0010000100_1000010001    // W0427_2048 = +0.257831    -0.966190
      0010000010_1000010001    // W0428_2048 = +0.254866    -0.966976
      0010000001_1000010001    // W0429_2048 = +0.251898    -0.967754
      0001111111_1000010000    // W0430_2048 = +0.248928    -0.968522
  40  0001111110_1000010000    // W0431_2048 = +0.245955    -0.969281
      0001111100_1000001111    // W0432_2048 = +0.242980    -0.970031
      0001111011_1000001111    // W0433_2048 = +0.240003    -0.970772
      0001111001_1000001111    // W0434_2048 = +0.237024    -0.971504
      0001111000_1000001110    // W0435_2048 = +0.234042    -0.972226
  45  0001110110_1000001110    // W0436_2048 = +0.231058    -0.972940
      0001110101_1000001101    // W0437_2048 = +0.228072    -0.973644
      0001110011_1000001101    // W0438_2048 = +0.225084    -0.974339
      0001110010_1000001101    // W0439_2048 = +0.222094    -0.975025
      0001110000_1000001100    // W0440_2048 = +0.219101    -0.975702
  50  0001101111_1000001100    // W0441_2048 = +0.216107    -0.976370
      0001101101_1000001100    // W0442_2048 = +0.213110    -0.977028
      0001101100_1000001011    // W0443_2048 = +0.210112    -0.977677
      0001101010_1000001011    // W0444_2048 = +0.207111    -0.978317
      0001101001_1000001011    // W0445_2048 = +0.204109    -0.978948
  55  0001100111_1000001010    // W0446_2048 = +0.201105    -0.979570
      0001100101_1000001010    // W0447_2048 = +0.198098    -0.980182
```

```
      0100000111_1001001001  // W0336_2048 = +0.514103   -0.857729
      0100000110_1001001000  // W0337_2048 = +0.511469   -0.859302
      0100000101_1001000111  // W0338_2048 = +0.508830   -0.860867
      0100000011_1001000110  // W0339_2048 = +0.506187   -0.862424
   5  0100000010_1001000110  // W0340_2048 = +0.503538   -0.863973
      0100000000_1001000101  // W0341_2048 = +0.500885   -0.865514
      0011111111_1001000100  // W0342_2048 = +0.498228   -0.867046
      0011111110_1001000011  // W0343_2048 = +0.495565   -0.868571
      0011111100_1001000011  // W0344_2048 = +0.492898   -0.870087
  10  0011111011_1001000010  // W0345_2048 = +0.490226   -0.871595
      0011111010_1001000001  // W0346_2048 = +0.487550   -0.873095
      0011111000_1001000000  // W0347_2048 = +0.484869   -0.874587
      0011110111_1000111111  // W0348_2048 = +0.482184   -0.876070
      0011110110_1000111111  // W0349_2048 = +0.479494   -0.877545
  15  0011110100_1000111110  // W0350_2048 = +0.476799   -0.879012
      0011110011_1000111101  // W0351_2048 = +0.474100   -0.880471
      0011110001_1000111100  // W0352_2048 = +0.471397   -0.881921
      0011110000_1000111100  // W0353_2048 = +0.468689   -0.883363
      0011101111_1000111011  // W0354_2048 = +0.465976   -0.884797
  20  0011101101_1000111010  // W0355_2048 = +0.463260   -0.886223
      0011101100_1000111010  // W0356_2048 = +0.460539   -0.887640
      0011101010_1000111001  // W0357_2048 = +0.457813   -0.889048
      0011101001_1000111000  // W0358_2048 = +0.455084   -0.890449
      0011101000_1000110111  // W0359_2048 = +0.452350   -0.891841
  25  0011100110_1000110111  // W0360_2048 = +0.449611   -0.893224
      0011100101_1000110110  // W0361_2048 = +0.446869   -0.894599
      0011100011_1000110101  // W0362_2048 = +0.444122   -0.895966
      0011100010_1000110101  // W0363_2048 = +0.441371   -0.897325
      0011100001_1000110100  // W0364_2048 = +0.438616   -0.898674
  30  0011011111_1000110011  // W0365_2048 = +0.435857   -0.900016
      0011011110_1000110011  // W0366_2048 = +0.433094   -0.901349
      0011011100_1000110010  // W0367_2048 = +0.430326   -0.902673
      0011011011_1000110001  // W0368_2048 = +0.427555   -0.903989
      0011011001_1000110000  // W0369_2048 = +0.424780   -0.905297
  35  0011011000_1000110000  // W0370_2048 = +0.422000   -0.906596
      0011010111_1000101111  // W0371_2048 = +0.419217   -0.907886
      0011010101_1000101111  // W0372_2048 = +0.416430   -0.909168
      0011010100_1000101110  // W0373_2048 = +0.413638   -0.910441
      0011010010_1000101101  // W0374_2048 = +0.410843   -0.911706
  40  0011010001_1000101101  // W0375_2048 = +0.408044   -0.912962
      0011001111_1000101100  // W0376_2048 = +0.405241   -0.914210
      0011001110_1000101011  // W0377_2048 = +0.402435   -0.915449
      0011001101_1000101011  // W0378_2048 = +0.399624   -0.916679
      0011001011_1000101010  // W0379_2048 = +0.396810   -0.917901
  45  0011001010_1000101001  // W0380_2048 = +0.393992   -0.919114
      0011001000_1000101001  // W0381_2048 = +0.391170   -0.920318
      0011000111_1000101000  // W0382_2048 = +0.388345   -0.921514
      0011000101_1000101000  // W0383_2048 = +0.385516   -0.922701
      0011000100_1000100111  // W0384_2048 = +0.382683   -0.923880
  50  0011000010_1000100110  // W0385_2048 = +0.379847   -0.925049
      0011000001_1000100110  // W0386_2048 = +0.377007   -0.926210
      0011000000_1000100101  // W0387_2048 = +0.374164   -0.927363
      0010111110_1000100101  // W0388_2048 = +0.371317   -0.928506
      0010111101_1000100100  // W0389_2048 = +0.368467   -0.929641
  55  0010111011_1000100011  // W0390_2048 = +0.365613   -0.930767
      0010111010_1000100011  // W0391_2048 = +0.362756   -0.931884
```

```
      01010011 10_1001111100   // WO280_2048 = +0.?53173   -0.?57209
      0101001101_1001111011   // WO281_2048 = +0.650847   -0.759209
      0101001100_1001111010   // WO282_2048 = +0.648514   -0.761202
      0101001011_1001111001   // WO283_2048 = +0.646176   -0.763188
 5    0101001010_1001111000   // WO284_2048 = +0.643832   -0.765167
      0101001000_1001110111   // WO285_2048 = +0.641481   -0.767139
      0101000111_1001110110   // WO286_2048 = +0.639124   -0.769103
      0101000110_1001110101   // WO287_2048 = +0.636762   -0.771061
      0101000101_1001110100   // WO288_2048 = +0.634393   -0.773010
10    0101000100_1001110011   // WO289_2048 = +0.632019   -0.774953
      0101000010_1001110010   // WO290_2048 = +0.629638   -0.776888
      0101000001_1001110001   // WO291_2048 = +0.627252   -0.778817
      0101000000_1001110000   // WO292_2048 = +0.624859   -0.780737
      0100111111_1001101111   // WO293_2048 = +0.622461   -0.782651
15    0100111101_1001101110   // WO294_2048 = +0.620057   -0.784557
      0100111100_1001101101   // WO295_2048 = +0.617647   -0.786455
      0100111011_1001101100   // WO296_2048 = +0.615232   -0.788346
      0100111010_1001101011   // WO297_2048 = +0.612810   -0.790230
      0100111001_1001101010   // WO298_2048 = +0.610383   -0.792107
20    0100110111_1001101001   // WO299_2048 = +0.607950   -0.793975
      0100110110_1001101001   // WO300_2048 = +0.605511   -0.795837
      0100110101_1001101000   // WO301_2048 = +0.603067   -0.797691
      0100110100_1001100111   // WO302_2048 = +0.600616   -0.799537
      0100110010_1001100110   // WO303_2048 = +0.598161   -0.801376
25    0100110001_1001100101   // WO304_2048 = +0.595699   -0.803208
      0100110000_1001100100   // WO305_2048 = +0.593232   -0.805031
      0100101110_1001100011   // WO306_2048 = +0.590760   -0.806848
      0100101101_1001100010   // WO307_2048 = +0.588282   -0.808656
      0100101100_1001100001   // WO308_2048 = +0.585798   -0.810457
30    0100101011_1001100000   // WO309_2048 = +0.583309   -0.812251
      0100101001_1001011111   // WO310_2048 = +0.580814   -0.814036
      0100101000_1001011110   // WO311_2048 = +0.578314   -0.815814
      0100100111_1001011101   // WO312_2048 = +0.575808   -0.817585
      0100100110_1001011100   // WO313_2048 = +0.573297   -0.819348
35    0100100100_1001011100   // WO314_2048 = +0.570781   -0.821103
      0100100011_1001011011   // WO315_2048 = +0.568259   -0.822850
      0100100010_1001011010   // WO316_2048 = +0.565732   -0.824589
      0100100000_1001011001   // WO317_2048 = +0.563199   -0.826321
      0100011111_1001011000   // WO318_2048 = +0.560662   -0.828045
40    0100011110_1001010111   // WO319_2048 = +0.558119   -0.829761
      0100011100_1001010110   // WO320_2048 = +0.555570   -0.831470
      0100011011_1001010101   // WO321_2048 = +0.553017   -0.833170
      0100011010_1001010101   // WO322_2048 = +0.550458   -0.834863
      0100011001_1001010100   // WO323_2048 = +0.547894   -0.836548
45    0100010111_1001010011   // WO324_2048 = +0.545325   -0.838225
      0100010110_1001010010   // WO325_2048 = +0.542751   -0.839894
      0100010101_1001010001   // WO326_2048 = +0.540171   -0.841555
      0100010011_1001010000   // WO327_2048 = +0.537587   -0.843208
      0100010010_1001001111   // WO328_2048 = +0.534998   -0.844854
50    0100010001_1001001111   // WO329_2048 = +0.532403   -0.846491
      0100001111_1001001110   // WO330_2048 = +0.529804   -0.848120
      0100001110_1001001101   // WO331_2048 = +0.527199   -0.849742
      0100001101_1001001100   // WO332_2048 = +0.524590   -0.851355
      0100001011_1001001011   // WO333_2048 = +0.521975   -0.852961
55    0100001010_1001001010   // WO334_2048 = +0.519356   -0.854558
      0100001001_1001001010   // WO335_2048 = +0.516732   -0.856147
```

```
     0110001100_1010111011   // W0224_2048 = +0.773010      -0.634393
     0110001011_1010111010   // W0225_2048 = +0.771061      -0.636762
     0110001010_1010111001   // W0226_2048 = +0.769103      -0.639124
     0110001001_1010111000   // W0227_2048 = +0.767139      -0.641481
  5  0110001000_1010110110   // W0228_2048 = +0.765167      -0.643832
     0110000111_1010110101   // W0229_2048 = +0.763188      -0.646176
     0110000110_1010110100   // W0230_2048 = +0.761202      -0.648514
     0110000101_1010110011   // W0231_2048 = +0.759209      -0.650847
     0110000100_1010110010   // W0232_2048 = +0.757209      -0.653173
 10  0110000011_1010110000   // W0233_2048 = +0.755201      -0.655493
     0110000010_1010101111   // W0234_2048 = +0.753187      -0.657807
     0110000001_1010101110   // W0235_2048 = +0.751165      -0.660114
     0110000000_1010101101   // W0236_2048 = +0.749136      -0.662416
     0101111111_1010101100   // W0237_2048 = +0.747101      -0.664711
 15  0101111101_1010101010   // W0238_2048 = +0.745058      -0.667000
     0101111100_1010101001   // W0239_2048 = +0.743008      -0.669283
     0101111011_1010101000   // W0240_2048 = +0.740951      -0.671559
     0101111010_1010100111   // W0241_2048 = +0.738887      -0.673829
     0101111001_1010100110   // W0242_2048 = +0.736817      -0.676093
 20  0101111000_1010100101   // W0243_2048 = +0.734739      -0.678350
     0101110111_1010100100   // W0244_2048 = +0.732654      -0.680601
     0101110110_1010100010   // W0245_2048 = +0.730563      -0.682846
     0101110101_1010100001   // W0246_2048 = +0.728464      -0.685084
     0101110100_1010100000   // W0247_2048 = +0.726359      -0.687315
 25  0101110011_1010011111   // W0248_2048 = +0.724247      -0.689541
     0101110010_1010011110   // W0249_2048 = +0.722128      -0.691759
     0101110001_1010011101   // W0250_2048 = +0.720003      -0.693971
     0101110000_1010011100   // W0251_2048 = +0.717870      -0.696177
     0101101110_1010011010   // W0252_2048 = +0.715731      -0.698376
 30  0101101101_1010011001   // W0253_2048 = +0.713585      -0.700569
     0101101100_1010011000   // W0254_2048 = +0.711432      -0.702755
     0101101011_1010010111   // W0255_2048 = +0.709273      -0.704934
     0101101010_1010010110   // W0256_2048 = +0.707107      -0.707107
     0101101001_1010010101   // W0257_2048 = +0.704934      -0.709273
 35  0101101000_1010010100   // W0258_2048 = +0.702755      -0.711432
     0101100111_1010010011   // W0259_2048 = +0.700569      -0.713585
     0101100110_1010010010   // W0260_2048 = +0.698376      -0.715731
     0101100100_1010010000   // W0261_2048 = +0.696177      -0.717870
     0101100011_1010001111   // W0262_2048 = +0.693971      -0.720003
 40  0101100010_1010001110   // W0263_2048 = +0.691759      -0.722128
     0101100001_1010001101   // W0264_2048 = +0.689541      -0.724247
     0101100000_1010001100   // W0265_2048 = +0.687315      -0.726359
     0101011111_1010001011   // W0266_2048 = +0.685084      -0.728464
     0101011110_1010001010   // W0267_2048 = +0.682846      -0.730563
 45  0101011100_1010001001   // W0268_2048 = +0.680601      -0.732654
     0101011011_1010001000   // W0269_2048 = +0.678350      -0.734739
     0101011010_1010000111   // W0270_2048 = +0.676093      -0.736817
     0101011001_1010000110   // W0271_2048 = +0.673829      -0.738887
     0101011000_1010000101   // W0272_2048 = +0.671559      -0.740951
 50  0101010111_1010000100   // W0273_2048 = +0.669283      -0.743008
     0101010110_1010000011   // W0274_2048 = +0.667000      -0.745058
     0101010100_1010000001   // W0275_2048 = +0.664711      -0.747101
     0101010011_1010000000   // W0276_2048 = +0.662416      -0.749136
     0101010010_1001111111   // W0277_2048 = +0.660114      -0.751165
 55  0101010001_1001111110   // W0278_2048 = +0.657807      -0.753187
     0101010000_1001111101   // W0279_2048 = +0.655493      -0.755201
```

```
      0110111101_1100000100  // W0168_2048 = +0.870087    -0.492898
      0110111101_1100000010  // W0169_2048 = +0.868571    -0.495565
      0110111100_1100000001  // W0170_2048 = +0.867046    -0.498228
      0110111011_1100000000  // W0171_2048 = +0.865514    -0.500885
 5    0110111010_1011111110  // W0172_2048 = +0.863973    -0.503538
      0110111010_1011111101  // W0173_2048 = +0.862424    -0.506187
      0110111001_1011111011  // W0174_2048 = +0.860867    -0.508830
      0110111000_1011111010  // W0175_2048 = +0.859302    -0.511469
      0110110111_1011111001  // W0176_2048 = +0.857729    -0.514103
10    0110110110_1011110111  // W0177_2048 = +0.856147    -0.516732
      0110110110_1011110110  // W0178_2048 = +0.854558    -0.519356
      0110110101_1011110101  // W0179_2048 = +0.852961    -0.521975
      0110110100_1011110011  // W0180_2048 = +0.851355    -0.524590
      0110110011_1011110010  // W0181_2048 = +0.849742    -0.527199
15    0110110010_1011110001  // W0182_2048 = +0.848120    -0.529804
      0110110001_1011101111  // W0183_2048 = +0.846491    -0.532403
      0110110001_1011101110  // W0184_2048 = +0.844854    -0.534998
      0110110000_1011101101  // W0185_2048 = +0.843208    -0.537587
      0110101111_1011101011  // W0186_2048 = +0.841555    -0.540171
20    0110101110_1011101010  // W0187_2048 = +0.839894    -0.542751
      0110101101_1011101001  // W0188_2048 = +0.838225    -0.545325
      0110101100_1011100111  // W0189_2048 = +0.836548    -0.547894
      0110101011_1011100110  // W0190_2048 = +0.834863    -0.550458
      0110101011_1011100101  // W0191_2048 = +0.833170    -0.553017
25    0110101010_1011100100  // W0192_2048 = +0.831470    -0.555570
      0110101001_1011100010  // W0193_2048 = +0.829761    -0.558119
      0110101000_1011100001  // W0194_2048 = +0.828045    -0.560662
      0110100111_1011100000  // W0195_2048 = +0.826321    -0.563199
      0110100110_1011011110  // W0196_2048 = +0.824589    -0.565732
30    0110100101_1011011101  // W0197_2048 = +0.822850    -0.568259
      0110100100_1011011100  // W0198_2048 = +0.821103    -0.570781
      0110100100_1011011010  // W0199_2048 = +0.819348    -0.573297
      0110100011_1011011001  // W0200_2048 = +0.817585    -0.575808
      0110100010_1011011000  // W0201_2048 = +0.815814    -0.578314
35    0110100001_1011010111  // W0202_2048 = +0.814036    -0.580814
      0110100000_1011010101  // W0203_2048 = +0.812251    -0.583309
      0110011111_1011010100  // W0204_2048 = +0.810457    -0.585798
      0110011110_1011010011  // W0205_2048 = +0.808656    -0.588282
      0110011101_1011010010  // W0206_2048 = +0.806848    -0.590760
40    0110011100_1011010000  // W0207_2048 = +0.805031    -0.593232
      0110011011_1011001111  // W0208_2048 = +0.803208    -0.595699
      0110011010_1011001110  // W0209_2048 = +0.801376    -0.598161
      0110011001_1011001100  // W0210_2048 = +0.799537    -0.600616
      0110011000_1011001011  // W0211_2048 = +0.797691    -0.603067
45    0110010111_1011001010  // W0212_2048 = +0.795837    -0.605511
      0110010111_1011001001  // W0213_2048 = +0.793975    -0.607950
      0110010110_1011000111  // W0214_2048 = +0.792107    -0.610383
      0110010101_1011000110  // W0215_2048 = +0.790230    -0.612810
      0110010100_1011000101  // W0216_2048 = +0.788346    -0.615232
50    0110010011_1011000100  // W0217_2048 = +0.786455    -0.617647
      0110010010_1011000011  // W0218_2048 = +0.784557    -0.620057
      0110010001_1011000001  // W0219_2048 = +0.782651    -0.622461
      0110010000_1011000000  // W0220_2048 = +0.780737    -0.624859
      0110001111_1010111111  // W0221_2048 = +0.778817    -0.627252
55    0110001110_1010111110  // W0222_2048 = +0.776888    -0.629638
      0110001101_1010111100  // W0223_2048 = +0.774953    -0.632019
```

```
       0111100010_1101010100   // W0112_2048 = +0.941544    -0.336890
       0111100010_1101010010   // W0113_2048 = +0.940506    -0.339777
       0111100001_1101010001   // W0114_2048 = +0.939459    -0.342661
       0111100000_1101001111   // W0115_2048 = +0.938404    -0.345541
    5  0111100000_1101001110   // W0116_2048 = +0.937339    -0.348419
       0111011111_1101001100   // W0117_2048 = +0.936266    -0.351293
       0111011111_1101001011   // W0118_2048 = +0.935184    -0.354164
       0111011110_1101001001   // W0119_2048 = +0.934093    -0.357031
       0111011110_1101001000   // W0120_2048 = +0.932993    -0.359895
   10  0111011101_1101000110   // W0121_2048 = +0.931884    -0.362756
       0111011101_1101000101   // W0122_2048 = +0.930767    -0.365613
       0111011100_1101000011   // W0123_2048 = +0.929641    -0.368457
       0111011011_1101000010   // W0124_2048 = +0.928506    -0.371317
       0111011011_1101000000   // W0125_2048 = +0.927363    -0.374164
   15  0111011010_1100111111   // W0126_2048 = +0.926210    -0.377007
       0111011010_1100111110   // W0127_2048 = +0.925049    -0.379847
       0111011001_1100111100   // W0128_2048 = +0.923880    -0.382683
       0111011000_1100111011   // W0129_2048 = +0.922701    -0.385516
       0111011000_1100111001   // W0130_2048 = +0.921514    -0.388345
   20  0111010111_1100111000   // W0131_2048 = +0.920318    -0.391170
       0111010111_1100110110   // W0132_2048 = +0.919114    -0.393992
       0111010110_1100110101   // W0133_2048 = +0.917901    -0.396810
       0111010101_1100110011   // W0134_2048 = +0.916679    -0.399624
       0111010101_1100110010   // W0135_2048 = +0.915449    -0.402435
   25  0111010100_1100110001   // W0136_2048 = +0.914210    -0.405241
       0111010011_1100101111   // W0137_2048 = +0.912962    -0.408044
       0111010011_1100101110   // W0138_2048 = +0.911706    -0.410843
       0111010010_1100101100   // W0139_2048 = +0.910441    -0.413638
       0111010001_1100101011   // W0140_2048 = +0.909168    -0.416430
   30  0111010001_1100101001   // W0141_2048 = +0.907886    -0.419217
       0111010000_1100101000   // W0142_2048 = +0.906596    -0.422000
       0111010000_1100100111   // W0143_2048 = +0.905297    -0.424780
       0111001111_1100100101   // W0144_2048 = +0.903989    -0.427555
       0111001110_1100100100   // W0145_2048 = +0.902673    -0.430326
   35  0111001101_1100100010   // W0146_2048 = +0.901349    -0.433094
       0111001101_1100100001   // W0147_2048 = +0.900016    -0.435857
       0111001100_1100011111   // W0148_2048 = +0.898674    -0.438616
       0111001011_1100011110   // W0149_2048 = +0.897325    -0.441371
       0111001011_1100011101   // W0150_2048 = +0.895966    -0.444122
   40  0111001010_1100011011   // W0151_2048 = +0.894599    -0.446869
       0111001001_1100011010   // W0152_2048 = +0.893224    -0.449611
       0111001001_1100011000   // W0153_2048 = +0.891841    -0.452350
       0111001000_1100010111   // W0154_2048 = +0.890449    -0.455084
       0111000111_1100010110   // W0155_2048 = +0.889048    -0.457813
   45  0111000110_1100010100   // W0156_2048 = +0.887640    -0.460539
       0111000110_1100010011   // W0157_2048 = +0.886223    -0.463260
       0111000101_1100010001   // W0158_2048 = +0.884797    -0.465976
       0111000100_1100010000   // W0159_2048 = +0.883363    -0.468689
       0111000100_1100001111   // W0160_2048 = +0.881921    -0.471397
   50  0111000011_1100001101   // W0161_2048 = +0.880471    -0.474100
       0111000010_1100001100   // W0162_2048 = +0.879012    -0.476799
       0111000001_1100001010   // W0163_2048 = +0.877545    -0.479494
       0111000001_1100001001   // W0164_2048 = +0.876070    -0.482184
       0111000000_1100001000   // W0165_2048 = +0.874587    -0.484869
   55  0110111111_1100000110   // W0166_2048 = +0.873095    -0.487550
       0110111110_1100000101   // W0167_2048 = +0.871595    -0.490226
```

```
        01111111000_1_0101000    // W0056_2048 = +0.985278    -0.170962
        01111111000_1_0100111    // W0057_2048 = +0.984749    -0.173984
        01111111000_1110100101   // W0058_2048 = +0.984210    -0.177004
        01111111000_1110100100   // W0059_2048 = +0.983662    -0.180023
   5    01111110111_1110100010   // W0060_2048 = +0.983105    -0.183040
        01111110111_1110100001   // W0061_2048 = +0.982539    -0.186055
        01111110111_1110011111   // W0062_2048 = +0.981964    -0.189069
        01111110110_1110011110   // W0063_2048 = +0.981379    -0.192080
        01111110110_1110011100   // W0064_2048 = +0.980785    -0.195090
  10    01111110110_1110011011   // W0065_2048 = +0.980182    -0.198098
        01111110110_1110011001   // W0066_2048 = +0.979570    -0.201105
        01111110101_1110010111   // W0067_2048 = +0.978948    -0.204109
        01111110101_1110010110   // W0068_2048 = +0.978317    -0.207111
        01111110101_1110010100   // W0069_2048 = +0.977677    -0.210112
  15    01111110100_1110010011   // W0070_2048 = +0.977028    -0.213110
        01111110100_1110010001   // W0071_2048 = +0.976370    -0.216107
        01111110100_1110010000   // W0072_2048 = +0.975702    -0.219101
        01111110011_1110001110   // W0073_2048 = +0.975025    -0.222094
        01111110011_1110001101   // W0074_2048 = +0.974339    -0.225084
  20    01111110011_1110001011   // W0075_2048 = +0.973644    -0.228072
        01111110010_1110001010   // W0076_2048 = +0.972940    -0.231058
        01111110010_1110001000   // W0077_2048 = +0.972226    -0.234042
        01111110001_1110000111   // W0078_2048 = +0.971504    -0.237024
        01111110001_1110000101   // W0079_2048 = +0.970772    -0.240003
  25    01111110001_1110000100   // W0080_2048 = +0.970031    -0.242980
        01111110000_1110000010   // W0081_2048 = +0.969281    -0.245955
        01111110000_1110000001   // W0082_2048 = +0.968522    -0.248928
        01111101111_1101111111   // W0083_2048 = +0.967754    -0.251898
        01111101111_1101111110   // W0084_2048 = +0.966976    -0.254866
  30    01111101111_1101111100   // W0085_2048 = +0.966190    -0.257831
        01111101110_1101111010   // W0086_2048 = +0.965394    -0.260794
        01111101110_1101111001   // W0087_2048 = +0.964590    -0.263755
        01111101101_1101110111   // W0088_2048 = +0.963776    -0.266713
        01111101101_1101110110   // W0089_2048 = +0.962953    -0.269668
  35    01111101101_1101110100   // W0090_2048 = +0.962121    -0.272621
        01111101100_1101110011   // W0091_2048 = +0.961280    -0.275572
        01111101100_1101110001   // W0092_2048 = +0.960431    -0.278520
        01111101011_1101110000   // W0093_2048 = +0.959572    -0.281465
        01111101011_1101101110   // W0094_2048 = +0.958703    -0.284408
  40    01111101010_1101101101   // W0095_2048 = +0.957826    -0.287347
        01111101010_1101101011   // W0096_2048 = +0.956940    -0.290285
        01111101001_1101101010   // W0097_2048 = +0.956045    -0.293219
        01111101001_1101101000   // W0098_2048 = +0.955141    -0.296151
        01111101001_1101100111   // W0099_2048 = +0.954228    -0.299080
  45    01111101000_1101100101   // W0100_2048 = +0.953306    -0.302006
        01111101000_1101100100   // W0101_2048 = +0.952375    -0.304929
        01111100111_1101100010   // W0102_2048 = +0.951435    -0.307850
        01111100111_1101100001   // W0103_2048 = +0.950486    -0.310767
        01111100110_1101011111   // W0104_2048 = +0.949528    -0.313682
  50    01111100110_1101011110   // W0105_2048 = +0.948561    -0.316593
        01111100101_1101011100   // W0106_2048 = +0.947586    -0.319502
        01111100101_1101011011   // W0107_2048 = +0.946601    -0.322408
        01111100100_1101011001   // W0108_2048 = +0.945607    -0.325310
        01111100100_1101011000   // W0109_2048 = +0.944605    -0.328210
  55    01111100011_1101010110   // W0110_2048 = +0.943593    -0.331106
        01111100011_1101010101   // W0111_2048 = +0.942573    -0.334000
```

```
       0111111111_0000000000   // W0000_2048 = +1.000000    -0.000000
       0111111111_1111111110   // W0001_2048 = +0.999995    -0.003068
       0111111111_1111111101   // W0002_2048 = +0.999981    -0.006136
       0111111111_1111111011   // W0003_2048 = +0.999958    -0.009204
 5     0111111111_1111111010   // W0004_2048 = +0.999925    -0.012272
       0111111111_1111111000   // W0005_2048 = +0.999882    -0.015339
       0111111111_1111110111   // W0006_2048 = +0.999831    -0.018407
       0111111111_1111110101   // W0007_2048 = +0.999769    -0.021474
       0111111111_1111110011   // W0008_2048 = +0.999699    -0.024541
10     0111111111_1111110010   // W0009_2048 = +0.999619    -0.027608
       0111111111_1111110000   // W0010_2048 = +0.999529    -0.030675
       0111111111_1111101111   // W0011_2048 = +0.999431    -0.033741
       0111111111_1111101101   // W0012_2048 = +0.999322    -0.036807
       0111111111_1111101100   // W0013_2048 = +0.999205    -0.039873
15     0111111111_1111101010   // W0014_2048 = +0.999078    -0.042938
       0111111111_1111101000   // W0015_2048 = +0.998941    -0.046003
       0111111111_1111100111   // W0016_2048 = +0.998795    -0.049068
       0111111111_1111100101   // W0017_2048 = +0.998640    -0.052132
       0111111111_1111100100   // W0018_2048 = +0.998476    -0.055195
20     0111111111_1111100010   // W0019_2048 = +0.998302    -0.058258
       0111111111_1111100001   // W0020_2048 = +0.998118    -0.061321
       0111111111_1111011111   // W0021_2048 = +0.997925    -0.064383
       0111111111_1111011101   // W0022_2048 = +0.997723    -0.067444
       0111111111_1111011100   // W0023_2048 = +0.997511    -0.070505
25     0111111111_1111011010   // W0024_2048 = +0.997290    -0.073565
       0111111110_1111011001   // W0025_2048 = +0.997060    -0.076624
       0111111110_1111010111   // W0026_2048 = +0.996820    -0.079682
       0111111110_1111010110   // W0027_2048 = +0.996571    -0.082740
       0111111110_1111010100   // W0028_2048 = +0.996313    -0.085797
30     0111111110_1111010011   // W0029_2048 = +0.996045    -0.088854
       0111111110_1111010001   // W0030_2048 = +0.995767    -0.091909
       0111111110_1111001111   // W0031_2048 = +0.995481    -0.094963
       0111111110_1111001110   // W0032_2048 = +0.995185    -0.098017
       0111111101_1111001100   // W0033_2048 = +0.994879    -0.101070
35     0111111101_1111001011   // W0034_2048 = +0.994565    -0.104122
       0111111101_1111001001   // W0035_2048 = +0.994240    -0.107172
       0111111101_1111001000   // W0036_2048 = +0.993907    -0.110222
       0111111101_1111000110   // W0037_2048 = +0.993564    -0.113271
       0111111101_1111000100   // W0038_2048 = +0.993212    -0.116319
40     0111111100_1111000011   // W0039_2048 = +0.992850    -0.119365
       0111111100_1111000001   // W0040_2048 = +0.992480    -0.122411
       0111111100_1111000000   // W0041_2048 = +0.992099    -0.125455
       0111111100_1110111110   // W0042_2048 = +0.991710    -0.128498
       0111111100_1110111101   // W0043_2048 = +0.991311    -0.131540
45     0111111011_1110111011   // W0044_2048 = +0.990903    -0.134581
       0111111011_1110111010   // W0045_2048 = +0.990485    -0.137620
       0111111011_1110111000   // W0046_2048 = +0.990058    -0.140658
       0111111011_1110110110   // W0047_2048 = +0.989622    -0.143695
       0111111010_1110110101   // W0048_2048 = +0.989177    -0.146730
50     0111111010_1110110011   // W0049_2048 = +0.988722    -0.149765
       0111111010_1110110010   // W0050_2048 = +0.988258    -0.152797
       0111111010_1110110000   // W0051_2048 = +0.987784    -0.155828
       0111111001_1110101111   // W0052_2048 = +0.987301    -0.158858
       0111111001_1110101101   // W0053_2048 = +0.986809    -0.161886
55     0111111001_1110101100   // W0054_2048 = +0.986308    -0.164913
       0111111001_1110101010   // W0055_2048 = +0.985798    -0.167938
```

```
fft_window       #(wordlength,
          r_wordlength,
           AddressSize,
          FIFO_L,
          FIFO_L_bits,
          FIFO_N,
          FIFO_n,
          FIFO_A,
          FIFO_A_bits,
          lu_AddressSize,
          delta,
          acquired_symbols,
          pos_threshold,
          t_offset_threshold,
            w_advance,
           sincint_latency,
           iqdemod_latency)
     window (.in_xr(i_data),
          .in_xi(q_data),
          .clk(clk),
          .nrst(nrst),
          .valid_in(valid_in),
          .valid_out(valid_win_2_fft),
          .in_resync(in_resync),
          .out_iqgi(out_iqgi),
          .out_sincgi(out_sincgi),
          .out_rx_guard(out_rx_guard),
          .out_acquired(out_acquired),
          .out_fft_window(out_fft_window),
          .enable_3_4(enable_3),
          .out_test(out_test),
          .track_ram_address(track_addr),
          .xri_tmp1(track_data_in),
          .xri_tmp5(track_data_out),
          .track_ram_rnotw(track_rnw),
          .track_ram_enable(track_ram_enable),
            .ram_addr(ram_addr),
             .ram_enable(ram_enable),
             .ram_rnotw(ram_rnotw),
          .ram10_in(ram10_in),   // To 1K x 24 bit RAM.
             .ram10_out(ram10_out), // From 1K x 24 bit RAM.
             .x1r_10(x1r_10),    // To FFT datapath (I).
             .x1i_10(x1i_10),    // To FFT datapath (Q).
             .z2r_10(z2r_10),    // From FFT datapath (I)
             .z2i_10(z2i_10),    // From FFT datapath (Q)
          .fft_ram_rnotw(ram_rnotw_fft_2_win),
          .fft_ram_enable(ram_enable_fft_2_win),
          .fft_ram_addr(ram_addr_fft_2_win));
endmodule


                        Listing 16
     // 2048 point FFT twiddle factor coefficients (Radix 4+2).
     // Coefficients stored as non-fractional 10 bit integers (scale 1 ).
     // Real Coefficient (cosine value) is coefficient high-byte.
     // Imaginary Coefficient (sine value) is coefficient low-byte.
```

```
                  .valid_in(valid_win_2_fft),// Input valid.
                .out_xr(i_out),      // FFT output data, I.
                 .out_xi(q_out),     // FFT output data, Q.
                  .out_ovf(out_ovf),     // Overflow flag.
  5            .enable_0(enable_0),
                 .enable_1(enable_1),
                 .enable_2(enable_2),
                 .enable_3(ram_rnotw_fft_2_win),
                  .valid_out(valid_out),
 10            .ram_address(ram_addr_fft_2_win),
                 .ram_enable(ram_enable_fft_2_win),
                 .address_rom3(rom3_addr),
                 .address_rom4(rom4_addr),


 15            // RAM input ports.
                 .z2r_4(ram4_in[wordlength-1:0]),
                 .z2i_4(ram4_in[wordlength*2-1:wordlength]),
                 .z2r_5(ram5_in[wordlength-1:0]),
                 .z2i_5(ram5_in[wordlength*2-1:wordlength]),
 20            .z2r_6(ram6_in[wordlength-1:0]),
                 .z2i_6(ram6_in[wordlength*2-1:wordlength]),
                 .z2r_7(ram7_in[wordlength-1:0]),
                 .z2i_7(ram7_in[wordlength*2-1:wordlength]),
                 .z2r_8(ram8_in[wordlength-1:0]),
 25            .z2i_8(ram8_in[wordlength*2-1:wordlength]),
                 .z2r_9(ram9_in[wordlength-1:0]),
                 .z2i_9(ram9_in[wordlength*2-1:wordlength]),
                 .z2r_10(z2r_10),// Frm FFT datapath to window (I).
                 .z2i_10(z2i_10),// Frm FFT datapath to window (Q).
 30
               // RAM output ports.
                 .x1r_4(ram4_out[wordlength-1:0]),
                 .x1i_4(ram4_out[wordlength*2-1:wordlength]),
                 .x1r_5(ram5_out[wordlength-1:0]),
 35            .x1i_5(ram5_out[wordlength*2-1:wordlength]),
                 .x1r_6(ram6_out[wordlength-1:0]),
                 .x1i_6(ram6_out[wordlength*2-1:wordlength]),
                 .x1r_7(ram7_out[wordlength-1:0]),
                 .x1i_7(ram7_out[wordlength*2-1:wordlength]),
 40            .x1r_8(ram8_out[wordlength-1:0]),
                 .x1i_8(ram8_out[wordlength*2-1:wordlength]),
                 .x1r_9(ram9_out[wordlength-1:0]),
                 .x1i_9(ram9_out[wordlength*2-1:wordlength]),
                 .x1r_10(x1r_10),// To FFT datapath frm window (I).
 45            .x1i_10(x1i_10),// To FFT datapath frm window (Q).

               // ROM output ports.
                 .br_3(rom3_data[c_wordlength*2-1:c_wordlength]),
                 .bi_3(rom3_data[c_wordlength-1:0]),
 50            .br_4(rom4_data[c_wordlength*2-1:c_wordlength]),
                 .bi_4(rom4_data[c_wordlength-1:0]));


       // ----------------------------------------------------------------
       //        Instance FFT window processor.
 55    // ----------------------------------------------------------------
```

```
      wire [AddressSize-1:0] ram_addr,        // RAM address bus.
            ram_addr_fft_2_win;

      wire      clk,
 5            nrst,
              in_2k8k,
            in_resync,
            valid_in,
              out_ovf,
10             enable_0,
              enable_1,
              enable_2,
              enable_3,
            valid_out,
15            ram_enable,        // RAM enable signal.
              ram_rnotw,
            valid_win_2_fft,
              ram_rnotw_fft_2_win,
              ram_enable_fft_2_win,
20          track_rnw,
            track_ram_enable,
              out_iqgi,
            out_sincgi;

25    wire [wordlength-1:0] x1r_10, x1i_10,
            z2r_10, z2i_10;

      wire [wordlength-3:0] track_data_in,
            track_data_out;
30
      wire [FIFO_L_bits-1:0] track_addr;

      wire [1:0]     out_rx_guard;      // Determined guard.

35    wire [c_wordlength*2-1:0] rom3_data,
            rom4_data;

      wire [rom_AddressSize-6:0] rom3_addr;
      wire [rom_AddressSize-4:0] rom4_addr;
40
      wire [14:0]     out_test;

      // ----------------------------------------------------------
      //        Instance FFT processor.
45    // ----------------------------------------------------------

      fft_r22sdf      #(wordlength,
            c_wordlength,
            AddressSize,
50          rom_AddressSize,
              mult_scale)
            fft (.in_xr(i_data),      // FFT input data, I.
            .in_xi(q_data),      // FFT input data, Q
            .clk(clk),        // Master clock.
55          .nrst(nrst),       // Power-up reset.
              .in_2k8k(in_2k8k),     // 2K active low.
```

```
output [wordlength-3:0] track_data_in;

output [wordlength*2-1:0] ram4_in,        // Couple the I/Q data
             ram5_in,       // outputs of each BF
             ram6_in,       // processor to their
             ram7_in,       // respective memory
             ram8_in,       // inputs.
             ram9_in,
                ram10_in;

output [AddressSize-1:0] ram_addr;          // RAM address bus.

output          out_ovf,       // Overflow flag.
             enable_0,       // Enable clock 0.
             enable_1,       // Enable clock 1.
             enable_2,       // Enable clock 2.
             enable_3,       // Enable clock 3.
               valid_out,       // Output data valid.
               ram_enable,       // RAM enable.
               ram_rnotw,
               track_rnw,
             track_ram_enable,
               out_iqgi,  .
             out_sincgi;

output [FIFO_L_bits-1:0] track_addr;

output [wordlength-1:0] i_out,        // FFT output data, I.
             q_out;       // FFT output data, Q.

// --------------------------------------------------------------
//        Wire/register declarations.
// --------------------------------------------------------------

wire [9:0]      i_data,       // FFT/WIN input I.
             q_data;       // FFT/WIN output Q.

wire [wordlength-1:0] i_out,       // FFT output data, I.
             q_out;       // FFT output data, Q.

wire [wordlength*2-1:0] ram4_in,
             ram5_in,
             ram6_in,
             ram7_in,
             ram8_in,
             ram9_in,
                ram10_in;

wire [wordlength*2-1:0] ram4_out,
             ram5_out,
               ram6_out,
             ram7_out,
               ram8_out,
               ram9_out,
               ram10_out;
```

```
           parameter      wordlength = 12;    // Data wordlength.
           parameter      c_wordlength = 10;   // Coeff wordlength.
           parameter      AddressSize = 13;    // Size of address bus.
           parameter      rom_AddressSize = 13;  // ROM address bus size.
    5      parameter      mult_scale = 3;     // Multiplier scalling:
                          // 1 = /4096, 2 = /2048,
                          // 3 = /1024, 4 = /512.
           parameter      r_wordlength = 10;   // ROM data wordlength.
           parameter      FIFO_L = 256;       // Tracking FIFO length.
   10      parameter      FIFO_L_bits = 8;     // Track FIFO addr bits
           parameter      FIFO_N = 64;        // Acc length S(i-j).
           parameter      FIFO_n = 64;        // Acc length S(i-n-j).
           parameter      FIFO_A = 32;        // t_offset delay FIFO.
           parameter      FIFO_A_bits = 5;     // Track FIFO bits.
   15      parameter      lu_AddressSize = 15;  // log rom address size.
           parameter      delta = 20;         // Gu threshold distance
           parameter      acquired_symbols = 2;  // Acq symbls before trk
           parameter      pos_threshold = 3;   // for info only.
           parameter      t_offset_threshold = 10;  // t_offset valid thresh
   20      parameter      w_advance = 10;      // win trig frm boundary
           parameter      sincint_latency = 2;  // Latency to sinc intep
           parameter      iqdemod_latency = 168;  // Latency to IQ demod.

           // -----------------------------------------------------------------
   25      //           Input/Output ports.
           // -----------------------------------------------------------------

           input          clk,          // Master clock.
                          nrst,          // Power-up reset.
   30                     in_2k8k,        // 2K mode active low.
                          valid_in,       // Input data valid.
                          in_resync;

           input [9:0]    i_data,         // FFT input data, I.
   35                     q_data;         // FFT input data, Q.

           input [wordlength-3:0] track_data_out;

           input [wordlength*2-1:0] ram4_out,    // Couple the I/Q data
   40                     ram5_out,       // outputs from the
                          ram6_out,       // memory to the
                          ram7_out,       // respective butterfly
                          ram8_out,       // processors.
                          ram9_out,
   45                     ram10_out;

           input [c_wordlength*2-1:0] rom3_data,
                          rom4_data;

   50      output [rom_AddressSize-6:0] rom3_addr;
           output [rom_AddressSize-4:0] rom4_addr;

           output [14:0]  out_test;       // Temp testpin output.

   55      output [1:0]   out_rx_guard;    // Acquired gu length.
```

```
         Notes  :

         ***********************************************************************/

5
         `timescale 1ns / 100ps

         module fft_top      (i_data,
                   q_data,
10                  clk,
                    nrst,
                    in_resync,
                    in_2k8k,
                    valid_in,
15                  ram4_in,
                 ram5_in,
                 ram6_in,
                 ram7_in,
                 ram8_in,
20               ram9_in,
                    ram10_in,
                 i_out,
                   q_out,
                   out_ovf,
25               enable_0,
                   enable_1,
                   enable_2,
                   enable_3,
                 valid_out,
30               ram4_out,
                 ram5_out,
                   ram6_out,
                 ram7_out,
                   ram8_out,
35                 ram9_out,
                   ram10_out,
                   ram_addr,
                   ram_enable,
                   ram_rnotw,
40               rom3_addr,
                   rom4_addr,
                   rom3_data,
                   rom4_data,
                   track_addr,
45                 track_data_in,
                 track_data_out,
                   track_rnw,
                 track_ram_enable,
                   out_rx_guard,
50                 out_iqgi,
                 out_sincgi,
                   out_test);


         // ------------------------------------------------------------
55       //        Parameter definitions.
         // ------------------------------------------------------------
```

```
        t_offset_diff < (1 << 14)) //CORRECT TO DETECT via = 1 not 0
        )
      t_offset_valid <= 0;
    else
5     t_offset_valid <= 1;

    assign t_offset_thresh = (t_offset_valid) ? t_offset_diff : 0;

    // Setup FIFO to perform moving summation of t_offset values.
10  fft_sr_addr #(15, FIFO_A) sr_A (clk, t_offset_ctl,
              t_offset_thresh,    // Input.
              t_offset_dly);      // Output.

    // Compute the moving summation i.e t_offset(i-1) + t_offset(i-2) + ...
15  // We must NOT truncate or round acc as the error will grow across a symbol.
    always @(posedge clk)
      if (in_resync || !nrst)      // Clear accumulator at
        t_offset_avg <= 0;         // power-up or Resync.
      else if (t_offset_ctl)       // Wait until t_offset valid.
20      // Subtract as well as add when averager is full.
        t_offset_avg <= t_offset_avg + t_offset_thresh
                - ((fifo_a_add_sub) ? t_offset_dly : 0);

    assign t_offset_scaled =
25    {{(FIFO_A_bits){t_offset_avg[14]}},t_offset_avg[14:FIFO_A_bits]};

    // -----------------------------------------------------------
    //  Code to determine conditions for advancing/retarding tracking window.
    // -----------------------------------------------------------
30
    assign read_pos = t_offset_scaled;        // +ve (late) so
                    // delay read

    assign read_neg = 2047 + guard_length + 1 -      // -ve (early) so
35      (~t_offset_scaled + 1);          // advance read

    assign write_pos = guard_length + 1 +      // +ve (late) so
          t_offset_scaled;        // delay write

40  // PROBLEMS WHEN offset > guard_length + 1
    // (should not happen as we range check peaks in acq mode)
    assign write_neg = guard_length + 1 -        // -ve (early) so
          (~t_offset_scaled + 1);      // advance write

45  endmodule
```

Listing 15

```
// Sccsld: %W% %G%
50  /*****************************************************************************
        Copyright (c) 1997 Pioneer Digital Design Centre Limited

    Author   : Dawood Alam.

55  Description: Verilog code for a structural netlist coupling the Fast Fourier
        Transform (FFT) processor to the window acquisition hardware.
```

```
        if (guard_active < (2176+delta)&&      // Test for 2048+128
          guard_active > (2176-delta))         // pt guard length.
          begin
          out_rx_guard <= 2'b01;
5         guard_length <= 128;
          end

        if (guard_active < (2112+delta)&&      // Test for 2048+64
          guard_active > (2112-delta))         // pt guard length.
10        begin
          out_rx_guard <= 2'b00;
          guard_length <= 64;
          end

15      // Now two peaks per block situation (small guards)
        if (guard_active < (5120+delta)&&      // Test for 4096+512+512
          guard_active > (5120-delta))         // 512 pt guard length.
          begin
          out_rx_guard <= 2'b11;
20        guard_length <= 512;
          end

        if (guard_active < (4608+delta)&&      // Test for 4096+256+256
          guard_active > (4608-delta))         // 256 pt guard length.
25        begin
          out_rx_guard <= 2'b10;
          guard_length <= 256;
          end

30      if (guard_active < (4352+delta)&&      // Test for 4096+128+128
          guard_active > (4352-delta))         // 128 pt guard length.
          begin
          out_rx_guard <= 2'b01;
          guard_length <= 128;
35        end

        if (guard_active < (4224+delta)&&      // Test for 4096+64+64
          guard_active > (4224-delta))         // 64 pt guard length.
          begin
40        out_rx_guard <= 2'b00;
          guard_length <= 64;
          end
        end

45  // -----------------------------------------------------------
    //      Averager for t_offset in tracking mode.
    // -----------------------------------------------------------

    assign t_offset_diff = t_offset - (2*FIFO_N + FIFO_n); //dly 2 for latency?
50
    always @(posedge clk)
      if (in_resync || !nrst) //NEED TO ENABLE THIS!!!!!!
        t_offset_valid <= 0;
      else if ((t_offset_diff < (1 << 14 + 1) - t_offset_threshold && // Neg
55        t_offset_diff > (1 << 14 - 1)) ||
          (t_offset_diff > t_offset_threshold &&      // Pos
```

15

```
          if (read)            // Read if 'read'
          begin               // flag is set.
           if (read_address == FIFO_L-1)     // Stop read at
              begin           // end of FIFO.
5             read_address <= 0;
              read <= 1'b0;        // Clr read flag.
            end
          else
            read_address <= read_address + 1'b1;    // Inc r address.
10        end

       if (track_mode && t_count == guard_length+1)   // Write if the
          write <= 1'b1;              // read is guard
                       // depth into FIFO
15     if (write)
          begin
           if (write_address == FIFO_L-1)      // Stop write at
              begin            // end of FIFO.
               write_address <= 0;
20             write <= 1'b0;
            end
          else
              write_address <= write_address + 1'b1;    // Inc w address.
          end
25        end
       end

       always @(enable_1_4 or enable_3_4 or read or write or  // Assign read and
          read_address or write_address)       // write addresses
30     if (enable_3_4 && read)          // onto common
          track_ram_address = read_address;       // address bus
          else if (enable_1_4 && write)          // for tracking
          track_ram_address = write_address;       // tsyncram RAM.

35     // --------------------------------------------------------------
       //   Thresholding function to determine precise guard interval.
       // --------------------------------------------------------------

       always @(posedge clk)
40     if (enable_3_4 && guard_valid)
          begin
           // First, one peak per block situation (large guards)
           if (guard_active < (2560+delta)&&      // Test for 2048+512
              guard_active > (2560-delta))        // pt guard length.
45         begin
           out_rx_guard <= 2'b11;
           guard_length <= 512;
           end

50        if (guard_active < (2304+delta)&&      // Test for 2048+256
              guard_active > (2304-delta))        // pt guard length.
           begin
           out_rx_guard <= 2'b10;
           guard_length <= 256;
55         end
```

```
        if (in_resync || !nrst)           // Synchronous reset.
         out_sincgi <= 0;
        else if (enable_3_4 && tracking &&
            t_count == 15'd0 - sincint_latency)    // sincgi guard start.
 5       out_sincgi <= 1'b1;
        else if (enable_3_4 && tracking && // TO COMPLETE LATENCY STUFF
            t_count == sincint_latency) // sincgi guard stop.
         out_sincgi <= 1'b0;


 10     always @(posedge clk)           // Count over active
         if (in_resync || !nrst)        // interval to generate
          enable_fft <= 1'b0;           // FFT valid pulse.
         else if (enable_3_4 && tracking &&
            t_count == guard_length + FIFO_L/2 - w_advance) // FFT start point is
 15       enable_fft <= 1'b1;           // in middle of write
         else if (enable_3_4 && tracking &&        // into FIFO_L + advced.
            fft_valid_count == 2047) // FFT stop after 2048
          enable_fft <= 1'b0;           // samples.


 20     always @(posedge clk)
         if (in_resync || !nrst)          // Synchronous reset.
          fft_valid_count <= 0;
         else if (enable_3_4 && tracking && ~enable_fft) // Valid count = 0.
          fft_valid_count <= 0;          // until fft is enabled.
 25      else if (enable_3_4 && tracking && enable_fft)
          fft_valid_count <= fft_valid_count + 1'b1;    // Count when enabled.

        assign valid_out = enable_fft & valid_in;  //MUST SYNCHROS Vld every 3 clks?

 30     // ----------------------------------------------------------------
        //       Synchronous RAM address generators.
        // ----------------------------------------------------------------

        always @(posedge clk)           // Acqsition FIFO address gen.
 35      if (!nrst || in_resync)        // Synchronous reset.
          window_ram_addr <= 0;         // Address gen for acq mode.
         else if (enable_2_8)
          window_ram_addr <= window_ram_addr + 1'b1;

 40     assign ram_enable_8 = enable_2_8 || enable_3_8 ||
               enable_4_8 || enable_5_8;

        always @(posedge clk)           // Tracking FIFO address gen.
         begin
 45       if (!nrst || in_resync)
           begin
            read_address <= 0;          // Reset track FIFO read addr.
            write_address <= 0;         // Reset track FIFO write addr
            write <= 1'b0;              // Track FIFO, write disabled.
 50         read <= 1'b0;               // Track FIFO, read disabled.
           end
          else if (enable_3_4)
           begin
            if (track_mode && t_count == 0)        // Track FIFO read
 55          read <= 1'b1;              // trigger point.
```

```
              dpctl_reset <= 1'b0;        // dp ctl out of reset.
              if (read && f_ratio_valid)     // Peak detect on rd&vld
              begin
            if (f_ratio > max_peak &&
                f_ratio < (1 << r_wordlength)) // Is new peak larger?
                begin
                max_peak <= f_ratio;     // If so assign max_peak
                t_offset <= t_count;     // Store peak offset.
                  end
              if (read_address == FIFO_L-1) // If at end of FIFO_L
              begin           // move to next state.
                state <= track2;      // (read_Addr <> FIFO_L)
                max_peak <= 1'b0;     // Reset max peak value.
              end
              end
              else
                state <= track1;        // else wait in track1.
              end


/*S5*/ track2: begin
              if (read && f_ratio_valid)     // Peak detect on rd&vld
              begin
            if (f_ratio > max_peak &&
                f_ratio < (1 << r_wordlength)) // Is new peak larger?
                begin
                max_peak <= f_ratio;     // If so assign max_peak
                t_offset <= t_count;     // Store peak offset
                  end
              if (read_address == FIFO_L-1) // At end of FIFO_L
              begin           // move to next state.
                state <= track1;      // (read_Addr <> FIFO_L)
                max_peak <= 1'b0;     // Reset max peak value.
              end
              end
              else
                state <= track2;        // Wait in this state.
              end

        default:   state <= 3'bXXX;
      endcase


      // --------------------------------------------------------
      //          FFT window output decode logic.
      // --------------------------------------------------------

      always @(posedge clk)
        if (in_resync || !nrst)         // Synchronous reset.
          out_iqgi <= 0;
        else if (enable_3_4 && tracking &&
            t_count == 15'd0 - iqdemod_latency)   // iqgi guard start.
          out_iqgi <= 1'b1;
        else if (enable_3_4 && tracking &&
              t_count == iqdemod_latency) // iqgi guard stop.
          out_iqgi <= 1'b0;


      always @(posedge clk)
```

```
                        t_retime_acq <= 1'b1;
                      end
                    else              // Acquisition failed so
                      begin           // jump to start and
 5                    state <= start;       // increment the retry
                      retry <= retry + 1'b1;    // counter.
                      t_reset <= 1'b1;      // Reset t_count.
                      g_a_reset <= 1'b1;     // Reset g_a_count.
                      max_peak <= 1'b0;     // Reset maximum peak.
10                  end
                  end


      /*S3*/   peak3: begin
            t_retime_acq <= 1'b0;
15            g_a_reset <= 1'b0;        // Next block start cnt
            if (g_a_count < 2048+512)     // Search for pos peak2
              begin
            if (f_ratio > max_peak &&
                f_ratio < (1 << r_wordlength)) // Is new peak larger?
20              begin
                max_peak <= f_ratio;     // If so assign max_peak
                guard_active <= t_count;   // Assign guard_active.
                end
              end              // third block complete
25            else if(// First, one peak per block situation (large guards)
                (guard_active < (2048+guard_length // Peak test 2048
                    +delta)&&   // + guard length.
              guard_active > (2048+guard_length
                    -delta))||
30
              // Now two peaks per block situation (small guards)
              (guard_active < (4096+(2*guard_length)// Peak 4096 + 2
                    +delta)&&    //*guard length.
              guard_active > (4096+(2*guard_length)
35                  -delta)))
              begin
              acq_symbols <= acq_symbols+1'b1;// Another sym acqurd
                g_a_reset <= 1'b1;      // Reset g_a_count.
              max_peak <= 1'b0;     // Reset maximum peak.
40            t_retime_trk <= 1'b1;    // Retime t_count to trk
              track_mode <= 1'b1;     // Enter track mode.
            dpctl_reset <= 1'b1;     // Reset datapath count
                state <= track1;       // Enter track1 state.
                end
45            else              // Acquisition failed so
              begin             // jump to start and
              state <= start;        // increment the retry
              retry <= retry + 1'b1;    // counter.
              t_reset <= 1'b1;       // Reset t_count.
50            g_a_reset <= 1'b1;     // Reset g_a_count.
              max_peak <= 1'b0;     // Reset maximum peak.
              end
            end


55    /*S4*/ track1: begin
            t_retime_trk <= 1'b0;       // t_count out retime.
```

```
              f_ratio < (1 << r_wordlength)) // Is new peak larger?
              begin
              max_peak <= f_ratio;      // If so assign max_peak
              t_reset <= 1;       // Reset timing counter.
5               end
            end
            else              // First block complete.
            begin
              t_reset <= 1'b0;          // t_count out of reset.
10             g_a_reset <= 1'b1;        // Reset g_a_count.
              max_peak <= 1'b0;         // Reset max peak value.
              state <= peak2;       // Next block search.
            end
          end

15
/*S2*/    peak2: begin
          g_a_reset <= 1'b0;          // Next block start cnt
            if (g_a_count < 2048+512)      // Search for pos peak2
            begin
20        if (f_ratio > max_peak &&
              f_ratio < (1 << r_wordlength)) // Is new peak larger?
               begin
               max_peak <= f_ratio;      // If so assign max_peak
               guard_active <= t_count;   // Assign guard_active.
25              end
            end               // Second block complete
            else if(// First, one peak per block situation (large guards)
               (guard_active < (2560+delta)&&  // Test for 2048+512
            guard_active > (2560-delta))|| // pt guard length.
30
               (guard_active < (2304+delta)&&  // Test for 2048+256
               guard_active > (2304-delta))|| // pt guard length.

               (guard_active < (2176+delta)&&  // Test for 2048+128
35              guard_active > (2176-delta))|| // pt guard length.

               (guard_active < (2112+delta)&&  // Test for 2048+64
                guard_active > (2112-delta))|| // pt guard length.

40             // Now two peaks per block situation (small guards)
               (guard_active < (5120+delta)&&  // Test 4096+512+512
            guard_active > (5120-delta))|| // pt guard length.

               (guard_active < (4608+delta)&&  // Test 4096+256+256
45              guard_active > (4608-delta))|| // pt guard length.

               (guard_active < (4352+delta)&&  // Test 4096+128+128
                guard_active > (4352-delta))|| // pt guard length.

50             (guard_active < (4224+delta)&&  // Test 4096+64+64
                guard_active > (4224-delta))) // pt guard length.
               begin
               state <= peak3;       // Next peak search.
                g_a_reset <= 1'b1;     // Reset g_a_count.
55             max_peak <= 1'b0;       // Reset maximum peak.
               guard_valid <= 1'b1;
```

```
      fifo_a_count <= fifo_a_count + 1'b1;    // and t_offset is valid.

      always @(posedge clk)           // Create pulse on entering
      if (enable_3_4)           // track 4 or track 5 to clk
      begin                // t_offset_ctl once per state
         if ((state == track1 &&        // transition. We need to
           old_state != track1) ||      // clock the averager only
           (state == track2 &&        // once on entering state 4 or
           old_state != track2))       // state 5 hence t_offset_ctl
           pulse <= 1'b1;          // is gated with pulse.
         else
           pulse <= 1'b0;
         old_state <= state;
      end

      always @(posedge clk)
      if (in_resync || !nrst)
      tracking <= 1'b0;          // Read from 2K/8K FIFO first.
      else if (enable_3_4 && track_mode
         && dp_count == FIFO_L+1)     // Check if FIFO_L full in trk
      tracking <= 1'b1;           // then read tracking FIFO_L.


      // ------------------------------------------------------------
      //     FFT window timing and sync acquisition/tracking FSM
      // ------------------------------------------------------------

      always @(posedge clk)           // Acquisition mode FSM.
      if (in_resync || !nrst)        // Synchronous power-up reset.
      begin
         state <= start;          // FSM starts in resync.
         track_mode <= 1'b0;         // Start in acquisition mode.
         t_reset <= 1'b0;         // Reset main timing counter.
         dpctl_reset <= 1'b0;        // dp_ctl out of reset.
         g_a_reset <= 1'b0;         // Reset guard_active counter.
         max_peak <= 1'b0;         // Reset max peak value.
         retry <= 0;          // Reset no of retry's.
         acq_symbols <= 0;         // Reset acquired no symbols.
         guard_valid <= 1'b0;        // Guard data is valid.
         t_retime_acq <= 1'b0;        // Do not retime at resync.
         t_retime_trk <= 1'b0;        // Do not retime at resync.
      end
      else if (enable_3_4)
      case (state)
      /*S0*/   start: begin
         g_a_reset <= 1'b0;         // g_a_reset out of rst
           t_reset <= 1'b0;         // t_count out of reset.
           guard_valid <= 1'b0;        // Guard invalid.
           // MUST ACT ON RETRYS TOO!!
           state <= peak1;          // Enter peak1 state.
         end

      /*S1*/   peak1: begin
           t_reset <= 1'b0;         // t_count out of reset.
         if (g_a_count < 2048+512)     // Search for pos peak1
             begin
           if (f_ratio > max_peak &&
```

```
        fifo_a_add_sub <= 0;
        else if (enable_3_4 && fifo_a_count == FIFO_A)    // fifo_a is full
        fifo_a_add_sub <= 1;              // so add and sub.

5     always @(posedge clk)
        if (in_resync || !nrst)           // Synchronous reset.
          t_offset_avg_valid <= 1'b0;        // Average value is
        else if (enable_3_4 && fifo_a_count == FIFO_A + 1) // valid one cycle
          t_offset_avg_valid <= 1'b1;        // after add_sub sig.
10
        assign dp_control = enable_3_4 &&          // Datapath enable
              (~track_mode || track_mode && read); // in acq/track mode.

        assign t_offset_ctl = enable_3_4 && t_offset_valid  // clock averager
15              && pulse && !read && tracking;  // dp control signal.


        // -----------------------------------------------------------
        //   FFT window timing and sync acquisition/tracking timing counters.
        // -----------------------------------------------------------
20
        always @(posedge clk)
        if (in_resync   !nrst || t_reset)     // Synchronous power-up reset.
          t_count <= 0;              // Reset main timing counter.
        else if (enable_3_4 && t_retime_acq)     // Retime to count from last
25        t_count <= t_count - guard_active;    // peak to current time.
        else if (enable_3_4 && ~track_mode)     // Count if not in track mode
          t_count <= t_count + 1'b1;
        else if (enable_3_4 && t_retime_trk)     // Otherwise must be in track
          t_count <= t_count - guard_active     // so advance timing for acq
30              + (2*FIFO_N + FIFO_n + 2); // FIFO_L read trig point then
        else if (enable_3_4)
          begin              // wrap round t_count at
            if (t_count == 2047+guard_length)   // end of guard+active length.
              t_count <= 0;           // Needed as a reference to
35          else              // track peak movement in
              t_count <= t_count + 1'b1;     // capture window.
          end

        always @(posedge clk)
40      if (in_resync || !nrst || g_a_reset)     // Synchronous power-up reset.
          g_a_count <= 0;             // Reset guard_active counter.
          else if (enable_3_4 && f_ratio_valid)   // g_a count when f_ratio vald
          g_a_count <= g_a_count + 1'b1;      // Guard active timing counter

45      always @(posedge clk)        // Datapath timing counter.
        if (in_resync || !nrst || dpctl_reset)   // Synchronous reset.
          dp_count <= 0;             // Reset datapath control.
        else if (enable_3_4 && ~track_mode)     // Always count in acquire
          dp_count <= dp_count + 1'b1;      // mode on clk 0.
50      else if (enable_3_4 && track_mode && read) // Count when reading data in
          dp_count <= dp_count + 1'b1;      // tracking mode.

        always @(posedge clk)
        if (in_resync || !nrst)          // Synchronous reset.
55        fifo_a_count <= 0;
        else if (enable_3_4 && t_offset_ctl)    // Only clock averager if Trk
```

```
          acc <= acc + xri_tmp3 - ((acc_add_sub) ? xri_tmp4 : 0);

          assign lu_address = acc;        // Ensure lu_address is large enough to
                         // accomodate acc number range.
    5
          fft_window_lu #(r_wordlength, lu_AddressSize)    // Case table instance
          log_lu (clk, dp_control, lu_address, lu_data);    // for a log lookup.

          // Setup 5 bit FIFO to determine the delayed variance.
   10     fft_sr_addr #(r_wordlength, FIFO_n) sr_n (clk, dp_control, // Length=FIFO_n.
                      lu_data,      // Input.
                      xri_tmp9);    // Output.

          // Determine difference of logs and hence the f_ratio when it is valid.
   15     always @(lu_data or xri_tmp9 or f_ratio_valid)
           f_ratio = (f_ratio_valid) ? lu_data - xri_tmp9 : 1'b0;


          // -----------------------------------------------------------
          //      Positive threshold (for information only)
   20     // -----------------------------------------------------------

          assign pos_peak =((f_ratio >= pos_threshold &&
                f_ratio < (1 << r_wordlength)) ? 1'b1 : 1'b0);

   25     // -----------------------------------------------------------
          //      FFT window datapath control registers.
          // -----------------------------------------------------------

          always @(posedge clk)
   30      if (in_resync || !nrst || dpctl_reset)     // Synchronous reset.
            begin
              f_ratio_valid <= 1'b0;             // Initalise datapath
              acc_add <= 1'b0;                   // control registers.
              acc_add_sub <= 1'b0;
   35       end
          else if (enable_3_4 && ~read)          // Acquisition mode
            begin                                // Use 2K/8K FIFO.
              if (dp_count == 2047 + FIFO_N + FIFO_n + 1 + 1) // f_ratio only valid
               f_ratio_valid <= 1'b1;            // after sum of FIFO
   40         if (dp_count == 2047)              // +acc+ROM latencys
               acc_add <= 1'b1;                  // Add if acc full.
              if (dp_count == 2047+FIFO_N)       // Add/sub when FIFO
               acc_add_sub <= 1'b1;              // N is full.
            end
   45     else if (enable_3_4 && read)           // Tracking mode
            begin                                // Use FIFO_L.
              if (dp_count == FIFO_L + FIFO_N + FIFO_n + 1 + 1) // f_ratio only valid
               f_ratio_valid <= 1'b1;            // after sum of FIFO
              if (dp_count == FIFO_L)            // +acc+ROM latencys
   50         acc_add <= 1'b1;                   // Add if acc full.
              if (dp_count == FIFO_L + FIFO_N)   // Add/sub when FIFO
               acc_add_sub <= 1'b1;              // N is full.
            end

   55     always @(posedge clk)
           if (in_resync || !nrst)              // Synchronous reset.
```

```
begin                    // RAM Mux code.
  if (!tracking)              // In window acq
    begin                    // mode.
      msb_out = ram10_out[2*wordlength-1:wordlength];
      lsb_out = ram10_out[wordlength-1:0];        // Connect window
      ram10_in[2*wordlength-1:wordlength] = msb_in;   // datapath & RAM
      ram10_in[wordlength-1:0] = lsb_in;        // control signals
      ram_enable = ram_enable_8;
      ram_rnotw = enable_3_8;
      ram_addr = window_ram_addr;
    end
  else                     // In tracking
    begin                    // mode, therefore
      x1r_10 = ram10_out[2*wordlength-1:wordlength];   // FFT functional.
      x1i_10 = ram10_out[wordlength-1:0];
      ram10_in[2*wordlength-1:wordlength] = z2r_10;   // Connect FFT
      ram10_in[wordlength-1:0] = z2i_10;        // datapath & RAM
      ram_enable = fft_ram_enable;            // control signals
      ram_rnotw = fft_ram_rnotw;
      ram_addr = fft_ram_addr;
    end
end

assign track_ram_rnotw = enable_3_4 & read;
assign track_ram_enable = (enable_3_4 & read) || (enable_1_4 & write);

// Select which FIFO we read data from depending on tracking or acquire mode.
always @(xri_tmp5 or xri_tmp2 or tracking)
  if(tracking)
    xri_tmp6 = xri_tmp5;            // Tracking mode
  else                   // data.
    xri_tmp6 = xri_tmp2;            // Acquisition
                      // mode data.
// Perform computation of s(i-j)
always @(xri_tmp1 or xri_tmp6)
  xri_tmp7 = xri_tmp1 - xri_tmp6;

// Take the modulus of xri_tmp7;
always @(xri_tmp7)
  if (xri_tmp7[wordlength-2])          // Check MSB for
    xri_tmp3 = -xri_tmp7;            // neg number.
  else
    xri_tmp3 = xri_tmp7;

// Setup FIFO to perform moving summation of s(i-j) values.
fft_sr_addr #(wordlength-2, FIFO_N) sr_N (clk, dp_control, // Length=FIFO_N.
            xri_tmp3,   // Input.
            xri_tmp4);  // Output.

// Compute the moving summation i.e S(i-j) = s(i-1,j-1) + s(i-2,j-2) + ...
// We must NOT truncate or round acc as the error will grow across a symbol.
always @(posedge clk)
  if (in_resync || !nrst || dpctl_reset)    // Clear accumulator at
    acc <= 0;                 // power-up or Resync or trk.
  else if (dp_control & acc_add)        // Wait until acc data valid.
    // Subtract as well as add when 2K/8K FIFO is full.
```

```
      begin
        xr_reg <= in_xr;
        xi_reg <= in_xi;
      end
5     else if (enable_3_4)
      begin
        xr_reg <= in_xr;
        xi_reg <= in_xi;
      end
10
      // Take the modulus of in_xr and in_xi and add together (|in_xr| + |in_xi|).
      always @(xr_reg or xi_reg)
      begin
        if (xr_reg[wordlength-3])      // Checking MSB for negative number.
15        xr_tmp1 = -xr_reg;
        else
          xr_tmp1 = xr_reg;

        if (xi_reg[wordlength-3])      // Checking MSB for negative number.
20        xi_tmp1 = -xi_reg;
        else
          xi_tmp1 = xi_reg;

        xri_tmp1 = xr_tmp1 + xi_tmp1;
25    end

      assign even_symbol = r[2];

      always @(even_symbol or msb_out_tmp or ram_in or lsb_out) // Mux MSB/LSB to
30    if (even_symbol)                // allow 1K RAM
        begin                          // to act as a 2K
          ram_out = lsb_out;            // FIFO, possible
          lsb_in_tmp = ram_in;          // since data
        end                    // bitwidth is 2b
35    else                     // bits wide in
        begin                    // the 1K RAM and
          ram_out = msb_out_tmp;        // only b bits are
          msb_in = ram_in;              // required in the
        end              // data path.
40
      always @(posedge clk)          // Delay even
      begin                // symbols by one
        if (enable_5_8)          // symbol so that
          lsb_in <= lsb_in_tmp;          // two symbols are
45    if (enable_7_8)          // written & read
          msb_out_tmp <= msb_out;          // to the ram.
      end

      assign xri_tmp2 = ram_out;        // Map RAM I/O
50    assign ram_in = xri_tmp1;        // to dp wires.

      always @(ram10_out or msb_in or lsb_in or z2r_10 or z2i_10
              or ram_enable_8 or enable_3_8
                or fft_ram_enable or fft_ram_rnotw
55              or window_ram_addr or fft_ram_addr
                or tracking)          // FFT/WINDOW FIFO
```

```
wire [wordlength-1:0] ram_in;

reg [wordlength-1:0] lsb_out,
            msb_out;

reg [wordlength-1:0] ram_out,
         msb_in,
            lsb_in;          .    · ·

wire [wordlength*2-1:0] ram10_out;
reg [wordlength*2-1:0] ram10_in;

reg [wordlength-1:0] x1r_10, x1i_10;
wire [wordlength-1:0] z2r_10, z2i_10;

wire [14:0]     out_test;
wire [14:0]     t_offset_diff,     // Actual +/- difference
            t_offset_thresh,     // Valid offset (maybe)
            t_offset_dly,      // Delayed of above.
            t_offset_scalled,     // Scalled to t_offset.
            read_pos,          // read trig, +ve offset
            read_neg,          // read trig, -ve offset
            write_pos,         // write trg, +ve offset
            write_neg;         // write trg, -ve offset

assign out_test = t_offset_diff;
// --------------------------------------------------------------
//      Fast 40 MHz clock decoder and valid_in control.
// --------------------------------------------------------------

always @(posedge clk)            .
 if (!nrst)             // Synchronous power-up reset.
  r <= 0;
 else if (valid_in)           // Count if input data valid.
  r <= r + 1'b1;          .

assign enable_0_4 = valid_in & (~r[1] & ~r[0]); // Gate valid_in with
assign enable_1_4 = valid_in & (~r[1] & r[0]); // decoded enable signals
assign enable_2_4 = valid_in & ( r[1] & ~r[0]); // to control all reg's.
assign enable_3_4 = valid_in & ( r[1] & r[0]); // Enables every 4 clk's

assign enable_1_8 = valid_in & (~r[2] & ~r[1] & r[0]);
assign enable_2_8 = valid_in & (~r[2] & r[1] & ~r[0]);
assign enable_3_8 = valid_in & (~r[2] & r[1] & r[0]);
assign enable_4_8 = valid_in & ( r[2] & ~r[1] & ~r[0]); // Enables every 8
assign enable_5_8 = valid_in & ( r[2] & ~r[1] & r[0]); // clk's
assign enable_6_8 = valid_in & ( r[2] & r[1] & ~r[0]);
assign enable_7_8 = valid_in & ( r[2] & r[1] & r[0]);

// --------------------------------------------------------------
// The entire data path incorporating the FIFO's, ROM and comparators.
// --------------------------------------------------------------

// Register the data inputs to the windowing module.
always @(posedge clk)
 if (in_resync || !nrst)
```

```
reg [2:0]      r;           // Clock decode counter.
reg [1:0]      out_rx_guard;     // Determined guard.
reg [r_wordlength:0] f_ratio;       // Statistical F ratio.
reg [10:0]    . fft_valid_count;   // Counts no of FFT vlds
reg [AddressSize-1:0] window_ram_addr,    // ram_address counter.
      ram_addr;
reg [14:0]     t_count,        // Window timing count.
      t_offset;       // Peak offset from t_ct
reg [14:0]     g_a_count;      // Guard_active counter.
reg [14:0]     dp_count;       // Datapath timing count
reg [14:0]     t_offset_avg;     // Averaged offset.
reg [2:0]      state,         // Acq/Track FSM state.
      old_state;      // Old tracking state.
reg [9:0]      guard_length;     // Thresholded guard len

reg [FIFO_A_bits:0]  fifo_a_count;    // Count till fifo_a ful
            // 1 bit more -> retime


reg [r_wordlength-1:0] max_peak;     // Maximum positive peak

reg [wordlength-1:0] msb_out_tmp,     // Temporary stores for
      lsb_in_tmp;      // even symbols to RAM.
wire [AddressSize-1:0] fft_ram_addr;     // From FFT RAM addr gen

wire      clk,          // Master clock.
      nrst,         // Power-up reset.
      enable_0_4,      // Clock enable 0 in 4.
      enable_1_4,      // Clock enable 1 in 4.
      enable_2_4,      // Clock enable 2 in 4.
      enable_3_4,      // Clock enable 3 in 4.
      enable_0_8,      // Clock enable 0 in 8.
      enable_1_8,      // Clock enable 1 in 8.
      enable_2_8,      // Clock enable 2 in 8.
      enable_3_8,      // Clock enable 3 in 8.
      enable_4_8,      // Clock enable 4 in 8.
      enable_5_8,      // Clock enable 5 in 8.
      enable_6_8,      // Clock enable 6 in 8.
      enable_7_8,      // Clock enable 7 in 8.
      ram_enable_8,     // Acq FIFO enable.
      track_ram_enable,   // Tracking RAM enable
      track_ram_rnotw,    // Tracking RAM rnotw.
      even_symbol,      // valid on even symbols
      in_resync,       // Resync to acqn mode.
      pos_peak,       // +ve peak, ref only!
      dp_control,      // Datapath acq/trk ctl.
      t_offset_ctl,      // Trk averager dp ctl.
      fft_ram_rnotw,
      fft_ram_enable;
wire [lu_AddressSize-1:0] lu_address;
wire [r_wordlength-1:0] lu_data,
      xri_tmp9;
wire [wordlength-3:0] xri_tmp2,
      xri_tmp4,
      xri_tmp5,
      in_q,
      out_q;
```

```
            ram_rnotw;

    output [FIFO_L_bits-1:0] track_ram_address;  // racking ram address

5   output [1:0]      out_rx_guard;      // Acquired gu length.

    output [AddressSize-1:0] ram_addr;

    output [wordlength-1:0] x1r_10, x1i_10;   // To FFT datapath.
10
    // -----------------------------------------------------------
    //        Wire/register declarations.
    // -----------------------------------------------------------

15  reg        out_acquired,      // Symbol acquired flag.
               out_fft_window,    // FFT window signal.
               tracking,          // Tracking mode data.
               acc_add,           // Acc add only flag.
               acc_add_sub,       // Acc add/sub flag.
20             fifo_a_add_sub,    // FIFO_A add/sub flag.
               f_ratio_valid,     // F ratio is valid
               read,              // Track FIFO read flag.
               write,             // Track FIFO write flag
               track_mode,        // Track/Acq status flag
25             dpctl_reset,       // Datapath control rst.
               t_reset,           // Timing counter reset.
               g_a_reset,         // Guard_active cnt rst.
               guard_valid,       // Guard signal is valid
               t_retime_acq,      // Retime timing counter
30             t_retime_trk,      // Retiming for tracking
               t_offset_valid,    // Peak offset valid.
               t_offset_avg_valid,  // Average offset valid.
               pulse,             // Pulse on states 4 & 5
               enable_fft,        // FFT enabled flag.
35             out_sincgi,        // Guard int to sincint.
               out_iqgi,          // Guard int to iq demod
               ram_enable,
               ram_rnotw;

40  reg [14:0]    guard_active;      // Guard+active length.
    reg [3:0]     retry,             // No failed retry's.
               acq_symbols;          // No of acquired symbls
    reg [wordlength-2:0] xri_tmp7;    // Delayed difference.
    reg [wordlength-3:0] xr_reg,       // (10 bits)
45          xi_reg,
               xri_tmp1,          // Sum of |I| + |Q|.
               xri_tmp3,          // Delayed |difference|.
               xri_tmp6;          // FIFO 2K/L output.

50  reg [FIFO_L_bits-1:0] read_address,    // Track FIFO read addr.
               write_address,     // Track FIFO write adr.
               track_ram_address;   // Tracking ram address;

    reg [lu_AddressSize-1:0] acc;       // Holds input variance.
55  reg [wordlength-4:0] xr_tmp1,     // |I|.
               xi_tmp1;           // |Q|.
```

```
parameter          FIFO_N = 64;        // Acc length S(i-j).
parameter          FIFO_n = 64;        // Acc length S(i-n-j).
parameter          FIFO_A = 32;        // t_offset dly FIFO+1.
parameter          FIFO_A_bits = 5;    // Track FIFO bits.
parameter          lu_AddressSize = 15;   // log lu address size.
parameter          delta = 20;         // Gu threshold distance
parameter          acquired_symbols = 2;   // Acq symbls before trk
parameter          pos_threshold = 3;  // For info only.
parameter          t_offset_threshold = 10; // t_offset valid thresh
parameter          w_advance = 10;     // win trig frm boundary
parameter          sincint_latency = 2;   // Latency to sinc intep
parameter          iqdemod_latency = 168;  // Latency to IQ demod.

parameter          start  = 3'b000,    // Search for neg peak.
                   peak1  = 3'b001,    // 1st pos peak found.
                   peak2  = 3'b010,    // 2nd pos peak found.
                   peak3  = 3'b011,    // 3rd pos peak found.
                   track1 = 3'b100,    // Tracking mode1.
                   track2 = 3'b101;    // Tracking mode1.

// --------------------------------------------------------------
//           Input/Output ports.
// --------------------------------------------------------------

input         clk,         // Master clock.
              nrst,        // Power-up reset.
              valid_in,    // Input data valid.
              in_resync,   // Sync FSM into Acqure.
              fft_ram_rnotw,
              fft_ram_enable;

input [AddressSize-1:0]  fft_ram_addr;

input [wordlength-3:0] in_xr,    // FFT input data, I.
              in_xi,      // FFT input data, Q.
              xri_tmp5;   // Track RAM output.

input [wordlength*2-1:0] ram10_out;    // From 1K x 24 bit RAM.

input [wordlength-1:0] z2r_10, z2i_10;    // From FFT datapath.

output [wordlength*2-1:0] ram10_in;    // To 1K x 24 bit RAM.

output [wordlength-3:0] xri_tmp1;    // Track RAM input.

output [14:0]     out_test;    // Temp testpin output.

output        out_iqgi,    // I/Q demod guard info.
              out_sincgi,  // Sinc int. guard info.
              out_acquired,   // Symbol acquired flag.
              out_fft_window,   // FFT processor st/stp
              enable_3_4,
              valid_out,
              track_ram_rnotw,
              track_ram_enable,
              ram_enable,
```

of valid_in and provides the necessary signals for the I/Q
demodulator, sync interpolator and error handler.

To DO: Check between successive symbol acquires for consistency
5   in timing.
         Window timing pulse
         tracking mode, filter peaks
         IQ and sync interpolator guard pulses.
         Override functions for timing.
10   Gain confidence by comparing symbol_acq vs retrys

***/

`timescale 1ns / 100ps
15
module fft_window (in_xr,
         in_xi,
            clk,
            nrst,
20   valid_in,
            valid_out,
            in_resync,
         out_iqgi,
            out_sincgi,
25   out_rx_guard,
            out_acquired,
            out_fft_window,
            enable_3_4,
            out_test,
30   track_ram_address,
            xri_tmp1,
            xri_tmp5,
            track_ram_rnotw,
            track_ram_enable,
35   ram_addr,
            ram_enable,
            ram_rnotw,
            ram10_in,
            ram10_out,
40   x1r_10,   // To FFT datapath (I).
            x1i_10,   // To FFT datapath (Q).
            z2r_10,   // From FFT datapath (I)
            z2i_10,   // From FFT datapath (Q)
            fft_ram_rnotw,   // From FFT addr gen.
45   fft_ram_enable,   // From FFT addr gen.
            fft_ram_addr);   // From FFT addr gen.

    // --
    //   Parameter definitions.
50 // --

    parameter   wordlength = 12;   // Data wordlength.
    parameter   r_wordlength = 10;   // ROM data wordlength.
    parameter   AddressSize = 13;   // Size of address bus.
55 parameter   FIFO_L = 256;   // Tracking FIFO length.
    parameter   FIFO_L_bits = 8;   // Track FIFO addr bits

```
13'b1i1011111110z : data_tmp1 = 10'b0111111100;
13'b1110111111110 : data_tmp1 = 10'b0111111100;
13'b1111000zzzzzz : data_tmp1 = 10'b0111111100;
13'b1111001000zzz : data_tmp1 = 10'b0111111100;
13'b111110010010zz : data_tmp1 = 10'b0111111100;
13'b111100100110z : data_tmp1 = 10'b0111111100;

13'b11110010z111z : data_tmp1 = 10'b0111111101;
13'b1111001z10zzz : data_tmp1 = 10'b0111111101;
13'b1111001z110zz : data_tmp1 = 10'b0111111101;
13'b1111001z1110z : data_tmp1 = 10'b0111111101;
13'b111100110zzzz : data_tmp1 = 10'b0111111101;
13'b111100111111z : data_tmp1 = 10'b0111111101;
13'b1111010zzzzzz : data_tmp1 = 10'b0111111101;
13'b111101100zzzz : data_tmp1 = 10'b0111111101;
13'b1111011010zzz : data_tmp1 = 10'b0111111101;

13'b1111011z11zzz : data_tmp1 = 10'b0111111110;
13'b111101110zzzz : data_tmp1 = 10'b0111111110;
13'b1111011110zzz : data_tmp1 = 10'b0111111110;
13'b1111100zzzzzz : data_tmp1 = 10'b0111111110;
13'b11111010zzzzz : data_tmp1 = 10'b0111111110;
13'b11111011000zz : data_tmp1 = 10'b0111111110;

13'b1111110110z1zz : data_tmp1 = 10'b0111111111;
13'b11111011z10zz : data_tmp1 = 10'b0111111111;
13'b11111z1110zzz : data_tmp1 = 10'b0111111111;
13'b11111z11111zz : data_tmp1 = 10'b0111111111;
13'b1111110zzzzzz : data_tmp1 = 10'b0111111111;
13'b11111110zzzzz : data_tmp1 = 10'b0111111111;
13'b111111110zzzz : data_tmp1 = 10'b0111111111;
13'b11111111110zz : data_tmp1 = 10'b0111111111;
default: data_tmp1 = 10'bxxxxxxxxxx;
endcase

always @(posedge clk)
 if (enable_3)
  data_tmp2 <= data_tmp1;

assign out_data = data_tmp2;

endmodule
```

                                  Listing 14

```
// SccsId: %W% %G%
/*****************************************************************************
     Copyright (c) 1997 Pioneer Digital Design Centre Limited

     Author   : Dawood Alam.

     Description: Verilog code for windowing algorithm to enable detection of the
        "active interval" of the COFDM symbol for guard values of:
           64, 128, 256, 512 and an active interval of 2048. (RTL)

     Notes   : This module generates the window signal for the FFT in the form
```

```
        13'b1110110110zzzz : data_tmp1 = 10'b0111110111;
        13'b11011011110zz : data_tmp1 = 10'b0111110111;
        13'b1101110zzzzzz : data_tmp1 = 10'b0111110111;

5       13'b1101111zzzzzz : data_tmp1 = 10'b0111111000;
        13'b11100000zzzzz : data_tmp1 = 10'b0111111000;
        13'b111000010zzzz : data_tmp1 = 10'b0111111000;
        13'b1110000110zzz : data_tmp1 = 10'b0111111000;
        13'b11100001110zz : data_tmp1 = 10'b0111111000;
10      13'b111000011110z : data_tmp1 = 10'b0111111000;

        13'b111000z11111z : data_tmp1 = 10'b0111111001;
        13'b11100010zzzzz : data_tmp1 = 10'b0111111001;
        13'b111000110zzzz : data_tmp1 = 10'b0111111001;
15      13'b1110001110zzz : data_tmp1 = 10'b0111111001;
        13'b11100011110zz : data_tmp1 = 10'b0111111001;
        13'b111000111110z : data_tmp1 = 10'b0111111001;
        13'b11100100zzzzz : data_tmp1 = 10'b0111111001;
        13'b111001010zzzz : data_tmp1 = 10'b0111111001;
20      13'b1110010110zzz : data_tmp1 = 10'b0111111001;
        13'b11100101110zz : data_tmp1 = 10'b0111111001;
        13'b111001011110z : data_tmp1 = 10'b0111111001;
        13'b1110010111110 : data_tmp1 = 10'b0111111001;

25      13'b111001z111111 : data_tmp1 = 10'b0111111010;
        13'b11100110zzzzz : data_tmp1 = 10'b0111111010;
        13'b111001110zzzz : data_tmp1 = 10'b0111111010;
        13'b1110011110zzz : data_tmp1 = 10'b0111111010;
        13'b11100111110zz : data_tmp1 = 10'b0111111010;
30      13'b111001111110z : data_tmp1 = 10'b0111111010;
        13'b1110011111110 : data_tmp1 = 10'b0111111010;
        13'b1110100zzzzzz : data_tmp1 = 10'b0111111010;
        13'b111010100000z : data_tmp1 = 10'b0111111010;

35      13'b11101010002z1z : data_tmp1 = 10'b0111111011;
        13'b111010100z10z : data_tmp1 = 10'b0111111011;
        13'b11101010z10zz : data_tmp1 = 10'b0111111011;
        13'b11101010z111z : data_tmp1 = 10'b0111111011;
        13'b1110101z10zzz : data_tmp1 = 10'b0111111011;
40      13'b1110101z1110z : data_tmp1 = 10'b0111111011;
        13'b111010110zzzz : data_tmp1 = 10'b0111111011;
        13'b11101011110zz : data_tmp1 = 10'b0111111011;
        13'b111010111111z : data_tmp1 = 10'b0111111011;
        13'b1110110zzzzzz : data_tmp1 = 10'b0111111011;
45      13'b11101110000zz : data_tmp1 = 10'b0111111011;
        13'b111011100010z : data_tmp1 = 10'b0111111011;
        13'b1110111000110 : data_tmp1 = 10'b0111111011;

        13'b1110111100z111 : data_tmp1 = 10'b0111111100;
50      13'b11101110z10zz : data_tmp1 = 10'b0111111100;
        13'b11101110z110z : data_tmp1 = 10'b0111111100;
        13'b11101110z1110 : data_tmp1 = 10'b0111111100;
        13'b1110111z10zzz : data_tmp1 = 10'b0111111100;
        13'b1110111z11111 : data_tmp1 = 10'b0111111100;
55      13'b111011110zzzz : data_tmp1 = 10'b0111111100;
        13'b11101111110zz : data_tmp1 = 10'b0111111100;
```

```
      13'b11000111111zzz : data_tmp1 = 10'b0111110010;
      13'b1100100zzzzzz : data_tmp1 = 10'b0111110010;
      13'b11001010zzzzz : data_tmp1 = 10'b0111110010;
      13'b1100101100zzz : data_tmp1 = 10'b0111110010;
  5   13'b110010110100z : data_tmp1 = 10'b0111110010;

      13'b1100101101z1z : data_tmp1 = 10'b0111110011;
      13'b11001011z110z : data_tmp1 = 10'b0111110011;
      13'b1100101110zzz : data_tmp1 = 10'b0111110011;
 10   13'b11001011110zz : data_tmp1 = 10'b0111110011;
      13'b110010111111z : data_tmp1 = 10'b0111110011;
      13'b1100110zzzzzz : data_tmp1 = 10'b0111110011;
      13'b110011100zzzz : data_tmp1 = 10'b0111110011;
      13'b1100111010zzz : data_tmp1 = 10'b0111110011;
 15   13'b11001110110zz : data_tmp1 = 10'b0111110011;
      13'b1100111011100 : data_tmp1 = 10'b0111110011;

      13'b11001110111z1 : data_tmp1 = 10'b0111110100;
      13'b1100111z11110 : data_tmp1 = 10'b0111110100;
 20   13'b110011110zzzz : data_tmp1 = 10'b0111110100;
      13'b1100111110zzz : data_tmp1 = 10'b0111110100;
      13'b11001111110zz : data_tmp1 = 10'b0111110100;
      13'b110011111110z : data_tmp1 = 10'b0111110100;
      13'b1100111111111 : data_tmp1 = 10'b0111110100;
 25   13'b1101000zzzzzz : data_tmp1 = 10'b0111110100;
      13'b110100100zzzz : data_tmp1 = 10'b0111110100;
      13'b110100101000z : data_tmp1 = 10'b0111110100;
      13'b1101001010010 : data_tmp1 = 10'b0111110100;

 30   13'b1101001010z11 : data_tmp1 = 10'b0111110101;
      13'b110100101z10z : data_tmp1 = 10'b0111110101;
      13'b110100101z110 : data_tmp1 = 10'b0111110101;
      13'b1101001z110zz : data_tmp1 = 10'b0111110101;
      13'b1101001z11111 : data_tmp1 = 10'b0111110101;
 35   13'b110100110zzzz : data_tmp1 = 10'b0111110101;
      13'b1101001110zzz : data_tmp1 = 10'b0111110101;
      13'b110100111110z : data_tmp1 = 10'b0111110101;
      13'b1101001111110 : data_tmp1 = 10'b0111110101;
      13'b1101010zzzzzz : data_tmp1 = 10'b0111110101;
 40   13'b1101011000zzz : data_tmp1 = 10'b0111110101;
      13'b110101100100z : data_tmp1 = 10'b0111110101;

      13'b1101011001z1z : data_tmp1 = 10'b0111110110;
      13'b11010110z110z : data_tmp1 = 10'b0111110110;
 45   13'b1101011z10zzz : data_tmp1 = 10'b0111110110;
      13'b1101011z110zz : data_tmp1 = 10'b0111110110;
      13'b1101011z1111z : data_tmp1 = 10'b0111110110;
      13'b110101110zzzz : data_tmp1 = 10'b0111110110;
      13'b110101111110z : data_tmp1 = 10'b0111110110;
 50   13'b1101100zzzzzz : data_tmp1 = 10'b0111110110;
      13'b11011010000zz : data_tmp1 = 10'b0111110110;

      13'b110110100z1zz : data_tmp1 = 10'b0111110111;
      13'b11011010z10zz : data_tmp1 = 10'b0111110111;
 55   13'b1101101z10zzz : data_tmp1 = 10'b0111110111;
      13'b1101101z111zz : data_tmp1 = 10'b0111110111;
```

```
      13'b1011100zzzzzz : data_tmp1 = 10'b0111101101;
      13'b101110100000zz : data_tmp1 = 10'b0111101101;
      13'b101110100010z : data_tmp1 = 10'b0111101101;

 5    13'b101110100z11z : data_tmp1 = 10'b0111101110;
      13'b10111010z10zz : data_tmp1 = 10'b0111101110;
      13'b10111010z110z : data_tmp1 = 10'b0111101110;
      13'b1011101z10zzz : data_tmp1 = 10'b0111101110;
      13'b1011101z1111z : data_tmp1 = 10'b0111101110;
10    13'b101110110zzzz : data_tmp1 = 10'b0111101110;
      13'b10111011110zz : data_tmp1 = 10'b0111101110;
      13'b101110111110z : data_tmp1 = 10'b0111101110;
      13'b10111100zzzzz : data_tmp1 = 10'b0111101110;
      13'b1011110100zzz : data_tmp1 = 10'b0111101110;
15    13'b10111101010zz : data_tmp1 = 10'b0111101110;
      13'b101111010110z : data_tmp1 = 10'b0111101110;
      13'b1011110101110 : data_tmp1 = 10'b0111101110;

      13'b10111101z1111 : data_tmp1 = 10'b0111101111;
20    13'b101111z110zzz : data_tmp1 = 10'b0111101111;
      13'b101111z1110zz : data_tmp1 = 10'b0111101111;
      13'b101111z11110z : data_tmp1 = 10'b0111101111;
      13'b101111z111110 : data_tmp1 = 10'b0111101111;
      13'b10111110zzzzz : data_tmp1 = 10'b0111101111;
25    13'b101111110zzzz : data_tmp1 = 10'b0111101111;
      13'b1011111111111 : data_tmp1 = 10'b0111101111;
      13'b110000000zzzz : data_tmp1 = 10'b0111101111;
      13'b1100000010zzz : data_tmp1 = 10'b0111101111;
      13'b110000001100z : data_tmp1 = 10'b0111101111;
30    13'b1100000011010 : data_tmp1 = 10'b0111101111;

      13'b1100000011z11 : data_tmp1 = 10'b0111110000;
      13'b1100000z1110z : data_tmp1 = 10'b0111110000;
      13'b1100000z11110 : data_tmp1 = 10'b0111110000;
35    13'b110000z10zzzz : data_tmp1 = 10'b0111110000;
      13'b110000z110zzz : data_tmp1 = 10'b0111110000;
      13'b110000z1110zz : data_tmp1 = 10'b0111110000;
      13'b110000z111111 : data_tmp1 = 10'b0111110000;
      13'b11000010zzzzz : data_tmp1 = 10'b0111110000;
40    13'b110000111110z : data_tmp1 = 10'b0111110000;
      13'b110000111110 : data_tmp1 = 10'b0111110000;
      13'b1100010000zzz : data_tmp1 = 10'b0111110000;
      13'b1100010001000 : data_tmp1 = 10'b0111110000;

45    13'b11000100010z1 : data_tmp1 = 10'b0111110001;
      13'b1100010001z10 : data_tmp1 = 10'b0111110001;
      13'b11000100z110z : data_tmp1 = 10'b0111110001;
      13'b11000100z1111 : data_tmp1 = 10'b0111110001;
      13'b1100010z10zzz : data_tmp1 = 10'b0111110001;
50    13'b1100010z110zz : data_tmp1 = 10'b0111110001;
      13'b1100010z11110 : data_tmp1 = 10'b0111110001;
      13'b110001z10zzzz : data_tmp1 = 10'b0111110001;
      13'b110001011110z : data_tmp1 = 10'b0111110001;
      13'b110001011111 : data_tmp1 = 10'b0111110001;
55    13'b11000110zzzzz : data_tmp1 = 10'b0111110001;
      13'b1100011110zzz : data_tmp1 = 10'b0111110001;
```

```
         13'b1010010010z11 : data_tmp1 = 10'b0111100111;
         13'b101001001z10z : data_tmp1 = 10'b0111100111;
         13'b101001001z110 : data_tmp1 = 10'b0111100111;
         13'b1010010z110zz : data_tmp1 = 10'b0111100111;
5        13'b1010010z11111 : data_tmp1 = 10'b0111100111;
         13'b101001z10zzzz : data_tmp1 = 10'b0111100111;
         13'b1010010110zzz : data_tmp1 = 10'b0111100111;
         13'b101001011110z : data_tmp1 = 10'b0111100111;
         13'b1010010111110 : data_tmp1 = 10'b0111100111;
10       13'b10100110zzzzz : data_tmp1 = 10'b0111100111;

         13'b101001111zzzz : data_tmp1 = 10'b0111101000;
         13'b101010Czzzzzz : data_tmp1 = 10'b0111101000;
         13'b101010100zzzz : data_tmp1 = 10'b0111101000;
15
         13'b1010101z1zzzz : data_tmp1 = 10'b0111101001;
         13'b101010110zzzz : data_tmp1 = 10'b0111101001;
         13'b10101100zzzzz : data_tmp1 = 10'b0111101001;
         13'b101011010zzzz : data_tmp1 = 10'b0111101001;
20
         13'b101011z11zzzz : data_tmp1 = 10'b0111101010;
         13'b10101110zzzzz : data_tmp1 = 10'b0111101010;
         13'b101011110zzzz : data_tmp1 = 10'b0111101010;
         13'b101100000zzzz : data_tmp1 = 10'b0111101010;
25       13'b101100001000z : data_tmp1 = 10'b0111101010;
         13'b1011000010010 : data_tmp1 = 10'b0111101010;

         13'b1011000010z11 : data_tmp1 = 10'b0111101011;
         13'b101100001z10z : data_tmp1 = 10'b0111101011;
30       13'b101100001z110 : data_tmp1 = 10'b0111101011;
         13'b1011000z110zz : data_tmp1 = 10'b0111101011;
         13'b1011000z11111 : data_tmp1 = 10'b0111101011;
         13'b101100z10zzzz : data_tmp1 = 10'b0111101011;
         13'b1011000110zzz : data_tmp1 = 10'b0111101011;
35       13'b101100011110z : data_tmp1 = 10'b0111101011;
         13'b1011000111110 : data_tmp1 = 10'b0111101011;
         13'b10110010zzzzz : data_tmp1 = 10'b0111101011;
         13'b10110011100zz : data_tmp1 = 10'b0111101011;
         13'b101100111010z : data_tmp1 = 10'b0111101011;
40       13'b1011001110110 : data_tmp1 = 10'b0111101011;

         13'b101100111z111 : data_tmp1 = 10'b0111101100;
         13'b10110011110zz : data_tmp1 = 10'b0111101100;
         13'b101100111110z : data_tmp1 = 10'b0111101100;
45       13'b1011001111110 : data_tmp1 = 10'b0111101100;
         13'b1011010zzzzzz : data_tmp1 = 10'b0111101100;
         13'b101101100zzzz : data_tmp1 = 10'b0111101100;
         13'b1011011010zzz : data_tmp1 = 10'b0111101100;
         13'b10110110110zz : data_tmp1 = 10'b0111101100;
50       13'b101101101110z : data_tmp1 = 10'b0111101100;

         13'b1011011z1111z : data_tmp1 = 10'b0111101101;
         13'b101101110zzzz : data_tmp1 = 10'b0111101101;
         13'b1011011110zzz : data_tmp1 = 10'b0111101101;
55       13'b10110111110zz : data_tmp1 = 10'b0111101101;
         13'b101101111110z : data_tmp1 = 10'b0111101101;
```

```
        13'b1001011z1110z : data_tmp1 = 10'b0111100010;
        13'b1001011z11110 : data_tmp1 = 10'b0111100010;
        13'b100101110zzzz : data_tmp1 = 10'b0111100010;
        13'b1001011110zzz : data_tmp1 = 10'b0111100010;
   5    13'b1001011111111 : data_tmp1 = 10'b0111100010;
        13'b10011000zzzzz : data_tmp1 = 10'b0111100010;
        13'b1001100100zzz : data_tmp1 = 10'b0111100010;
        13'b10011001010zz : data_tmp1 = 10'b0111100010;
        13'b1001100101100 : data_tmp1 = 10'b0111100010;
  10
        13'b10011001011z1 : data_tmp1 = 10'b0111100011;
        13'b10011001z1110 : data_tmp1 = 10'b0111100011;
        13'b100110z110zzz : data_tmp1 = 10'b0111100011;
        13'b100110z1110zz : data_tmp1 = 10'b0111100011;
  15    13'b100110z11110z : data_tmp1 = 10'b0111100011;
        13'b100110z111111 : data_tmp1 = 10'b0111100011;
        13'b10011010zzzzz : data_tmp1 = 10'b0111100011;
        13'b100110110zzzz : data_tmp1 = 10'b0111100011;
        13'b1001101111110 : data_tmp1 = 10'b0111100011;
  20    13'b10011100000zz : data_tmp1 = 10'b0111100011;

        13'b100111000z1zz : data_tmp1 = 10'b0111100100;
        13'b10011100z10zz : data_tmp1 = 10'b0111100100;
        13'b1001110z10zzz : data_tmp1 = 10'b0111100100;
  25    13'b1001110z111zz : data_tmp1 = 10'b0111100100;
        13'b100111010zzzz : data_tmp1 = 10'b0111100100;
        13'b10011101110zz : data_tmp1 = 10'b0111100100;
        13'b100111100zzzz : data_tmp1 = 10'b0111100100;
        13'b1001111010zzz : data_tmp1 = 10'b0111100100;
  30    13'b10011110110zz : data_tmp1 = 10'b0111100100;
        13'b1001111011100 : data_tmp1 = 10'b0111100100;

        13'b100111110111z1 : data_tmp1 = 10'b0111100101;
        13'b1001111z11110 : data_tmp1 = 10'b0111100101;
  35    13'b100111110zzzz : data_tmp1 = 10'b0111100101;
        13'b1001111110zzz : data_tmp1 = 10'b0111100101;
        13'b10011111110zz : data_tmp1 = 10'b0111100101;
        13'b100111111110z : data_tmp1 = 10'b0111100101;
        13'b1001111111111 : data_tmp1 = 10'b0111100101;
  40    13'b10100000zzzzz : data_tmp1 = 10'b0111100101;
        13'b101000010zzzz : data_tmp1 = 10'b0111100101;
        13'b10100001100zz : data_tmp1 = 10'b0111100101;
        13'b101000011010z : data_tmp1 = 10'b0111100101;
        13'b1010000110110 : data_tmp1 = 10'b0111100101;
  45
        13'b101000011z111 : data_tmp1 = 10'b0111100110;
        13'b101000z1110zz : data_tmp1 = 10'b0111100110;
        13'b101000z11110z : data_tmp1 = 10'b0111100110;
        13'b101000z111110 : data_tmp1 = 10'b0111100110;
  50    13'b10100010zzzzz : data_tmp1 = 10'b0111100110;
        13'b101000110zzzz : data_tmp1 = 10'b0111100110;
        13'b1010001110zzz : data_tmp1 = 10'b0111100110;
        13'b10100011111 11 : data_tmp1 = 10'b0111100110;
        13'b101001000zzzz : data_tmp1 = 10'b0111100110;
  55    13'b101001001000z : data_tmp1 = 10'b0111100110;
        13'b1010010010010 : data_tmp1 = 10'b0111100110;
```

```
      13'b1000101z1110z : data_tmp1 = 10'b0111011101;
      13'b1000101z11111 : data_tmp1 = 10'b0111011101;
      13'b100010110zzzz : data_tmp1 = 10'b0111011101;
      13'b10001011110zz : data_tmp1 = 10'b0111011101;
   5  13'b1000101111110 : data_tmp1 = 10'b0111011101;
      13'b1000110000zzz : data_tmp1 = 10'b0111011101;
      13'b10001100010zz : data_tmp1 = 10'b0111011101;
      13'b100011000110z : data_tmp1 = 10'b0111011101;
      13'b1000110001110 : data_tmp1 = 10'b0111011101;
  10
      13'b10001100z1111 : data_tmp1 = 10'b0111011110;
      13'b1000110z10zzz : data_tmp1 = 10'b0111011110;
      13'b1000110z110zz : data_tmp1 = 10'b0111011110;
      13'b1000110z1110z : data_tmp1 = 10'b0111011110;
  15  13'b1000110z11110 : data_tmp1 = 10'b0111011110;
      13'b100011010zzzz : data_tmp1 = 10'b0111011110;
      13'b1000110111111 : data_tmp1 = 10'b0111011110;
      13'b100011100zzzz : data_tmp1 = 10'b0111011110;
      13'b1000111010zzz : data_tmp1 = 10'b0111011110;
  20  13'b10001110110zz : data_tmp1 = 10'b0111011110;
      13'b100011101110z : data_tmp1 = 10'b0111011110;
      13'b1000111011110 : data_tmp1 = 10'b0111011110;

      13'b1000111z11111 : data_tmp1 = 10'b0111011111;
  25  13'b100011110zzzz : data_tmp1 = 10'b0111011111;
      13'b1000111110zzz : data_tmp1 = 10'b0111011111;
      13'b10001111110zz : data_tmp1 = 10'b0111011111;
      13'b100011111110z : data_tmp1 = 10'b0111011111;
      13'b1000111111110 : data_tmp1 = 10'b0111011111;
  30  13'b10010000zzzzz : data_tmp1 = 10'b0111011111;
      13'b100100010zzzz : data_tmp1 = 10'b0111011111;

      13'b100100z11zzzz : data_tmp1 = 10'b0111100000;
      13'b10010010zzzzz : data_tmp1 = 10'b0111100000;
  35  13'b100100110zzzz : data_tmp1 = 10'b0111100000;
      13'b100101000000z : data_tmp1 = 10'b0111100000;
      13'b1001010000010 : data_tmp1 = 10'b0111100000;

      13'b1001010000z11 : data_tmp1 = 10'b0111100001;
  40  13'b100101000z10z : data_tmp1 = 10'b0111100001;
      13'b100101000z110 : data_tmp1 = 10'b0111100001;
      13'b10010100z10zz : data_tmp1 = 10'b0111100001;
      13'b10010100z1111 : data_tmp1 = 10'b0111100001;
      13'b1001010z10zzz : data_tmp1 = 10'b0111100001;
  45  13'b1001010z1110z : data_tmp1 = 10'b0111100001;
      13'b1001010z11110 : data_tmp1 = 10'b0111100001;
      13'b100101010zzzz : data_tmp1 = 10'b0111100001;
      13'b10010101110zz : data_tmp1 = 10'b0111100001;
      13'b1001010111111 : data_tmp1 = 10'b0111100001;
  50  13'b100101100zzzz : data_tmp1 = 10'b0111100001;
      13'b10010110100zz : data_tmp1 = 10'b0111100001;
      13'b100101101010z : data_tmp1 = 10'b0111100001;
      13'b1001011010110 : data_tmp1 = 10'b0111100001;

  55  13'b100101101z111 : data_tmp1 = 10'b0111100010;
      13'b1001011z110zz : data_tmp1 = 10'b0111100010;
```

```
        13'b0111110z10zzz : data_tmp1 = 10'b0111010111;
        13'b0111110z1111z : data_tmp1 = 10'b0111010111;
        13'b011111010zzzz : data_tmp1 = 10'b0111010111;
        13'b01111101110zz : data_tmp1 = 10'b0111010111;
   5    13'b0111111011110z : data_tmp1 = 10'b0111010111;
        13'b01111111000zzz : data_tmp1 = 10'b0111010111;
        13'b0111111110010zz : data_tmp1 = 10'b0111010111;
        13'b01111111001100 : data_tmp1 = 10'b0111010111;

  10    13'b01111110011z1 : data_tmp1 = 10'b0111011000;
        13'b01111110z1110 : data_tmp1 = 10'b0111011000;
        13'b0111111z10zzz : data_tmp1 = 10'b0111011000;
        13'b0111111z110zz : data_tmp1 = 10'b0111011000;
        13'b0111111z1110z : data_tmp1 = 10'b0111011000;
  15    13'b0111111z11111 : data_tmp1 = 10'b0111011000;
        13'b011111110zzzz : data_tmp1 = 10'b0111011000;
        13'b0111111111110 : data_tmp1 = 10'b0111011000;
        13'b100000000zzzz : data_tmp1 = 10'b0111011000;
        13'b10000000100zz : data_tmp1 = 10'b0111011000;
  20
        13'b100000001z1zz : data_tmp1 = 10'b0111011001;
        13'b1000000z110zz : data_tmp1 = 10'b0111011001;
        13'b100000010zzzz : data_tmp1 = 10'b0111011001;
        13'b1000000110zzz : data_tmp1 = 10'b0111011001;
  25    13'b10000001111zz : data_tmp1 = 10'b0111011001;
        13'b100000100zzzz : data_tmp1 = 10'b0111011001;
        13'b1000001010zzz : data_tmp1 = 10'b0111011001;
        13'b10000010110zz : data_tmp1 = 10'b0111011001;
        13'b100000101110z : data_tmp1 = 10'b0111011001;
  30
        13'b1000001z1111z : data_tmp1 = 10'b0111011010;
        13'b100000110zzzz : data_tmp1 = 10'b0111011010;
        13'b1000001110zzz : data_tmp1 = 10'b0111011010;
        13'b10000011110zz : data_tmp1 = 10'b0111011010;
  35    13'b100000111110z : data_tmp1 = 10'b0111011010;
        13'b10000100zzzzz : data_tmp1 = 10'b0111011010;
        13'b1000010100zzz : data_tmp1 = 10'b0111011010;

        13'b10000101z1zzz : data_tmp1 = 10'b0111011011;
  40    13'b1000010110zzz : data_tmp1 = 10'b0111011011;
        13'b10000110zzzzz : data_tmp1 = 10'b0111011011;
        13'b100001110zzzz : data_tmp1 = 10'b0111011011;
        13'b10000111100zz : data_tmp1 = 10'b0111011011;

  45    13'b100001111z1zz : data_tmp1 = 10'b0111011100;
        13'b10000111110zz : data_tmp1 = 10'b0111011100;
        13'b1000100zzzzzz : data_tmp1 = 10'b0111011100;
        13'b1000101000000 : data_tmp1 = 10'b0111011100;

  50    13'b10001010000z1 : data_tmp1 = 10'b0111011101;
        13'b1000101000z10 : data_tmp1 = 10'b0111011101;
        13'b100010100z10z : data_tmp1 = 10'b0111011101;
        13'b100010100z111 : data_tmp1 = 10'b0111011101;
        13'b10001010z10zz : data_tmp1 = 10'b0111011101;
  55    13'b10001010z1110 : data_tmp1 = 10'b0111011101;
        13'b1000101z10zzz : data_tmp1 = 10'b0111011101;
```

```
      13'b01101110zzzzz : data_tmp1 = 10'b0111010000;
      13'b011011110zzzz : data_tmp1 = 10'b0111010000;
      13'b0110111110zzz : data_tmp1 = 10'b0111010000;

5     13'b01101111111zzz : data_tmp1 = 10'b0111010001;
      13'b01110000zzzzz : data_tmp1 = 10'b0111010001;
      13'b011100010zzzz : data_tmp1 = 10'b0111010001;
      13'b01110001100zz : data_tmp1 = 10'b0111010001;
      13'b011100011010z : data_tmp1 = 10'b0111010001;
10    13'b0111000110110 : data_tmp1 = 10'b0111010001;

      13'b0111000011z111 : data_tmp1 = 10'b0111010010;
      13'b011100001110zz : data_tmp1 = 10'b0111010010;
      13'b011100011110z : data_tmp1 = 10'b0111010010;
15    13'b0111000111110 : data_tmp1 = 10'b0111010010;
      13'b01110010zzzzz : data_tmp1 = 10'b0111010010;
      13'b011100110zzzz : data_tmp1 = 10'b0111010010;
      13'b0111001110zzz : data_tmp1 = 10'b0111010010;

20    13'b01110011111zzz : data_tmp1 = 10'b0111010011;
      13'b01110100zzzzz : data_tmp1 = 10'b0111010011;
      13'b011101010zzzz : data_tmp1 = 10'b0111010011;
      13'b0111010110zzz : data_tmp1 = 10'b0111010011;
      13'b01110101011100z : data_tmp1 = 10'b0111010011;
25
      13'b011101010111z1z : data_tmp1 = 10'b0111010100;
      13'b011101011110z : data_tmp1 = 10'b0111010100;
      13'b01110110zzzzz : data_tmp1 = 10'b0111010100;
      13'b011101110zzzz : data_tmp1 = 10'b0111010100;
30    13'b0111011110zzz : data_tmp1 = 10'b0111010100;
      13'b01110111110zz : data_tmp1 = 10'b0111010100;
      13'b0111011111100 : data_tmp1 = 10'b0111010100;

      13'b011101111111z1 : data_tmp1 = 10'b0111010101;
35    13'b0111011111110 : data_tmp1 = 10'b0111010101;
      13'b0111100zzzzzz : data_tmp1 = 10'b0111010101;
      13'b0111101000000 : data_tmp1 = 10'b0111010101;

      13'b011111010000z1 : data_tmp1 = 10'b0111010110;
40    13'b0111101000z10 : data_tmp1 = 10'b0111010110;
      13'b011110100z10z : data_tmp1 = 10'b0111010110;
      13'b011110100z111 : data_tmp1 = 10'b0111010110;
      13'b01111010z10zz : data_tmp1 = 10'b0111010110;
      13'b01111010z1110 : data_tmp1 = 10'b0111010110;
45    13'b0111101z10zzz : data_tmp1 = 10'b0111010110;
      13'b0111101z1110z : data_tmp1 = 10'b0111010110;
      13'b0111101z11111 : data_tmp1 = 10'b0111010110;
      13'b011110110zzzz : data_tmp1 = 10'b0111010110;
      13'b01111011110zz : data_tmp1 = 10'b0111010110;
50    13'b0111101111110 : data_tmp1 = 10'b0111010110;
      13'b01111100000zz : data_tmp1 = 10'b0111010110;
      13'b011111000010z : data_tmp1 = 10'b0111010110;

      13'b011111000z11z : data_tmp1 = 10'b0111010111;
55    13'b01111100z10zz : data_tmp1 = 10'b0111010111;
      13'b01111100z110z : data_tmp1 = 10'b0111010111;
```

```
    13'b0110010000zzzz : data_tmp1 = 10'b0111001010;
    13'b01100100010000 : data_tmp1 = 10'b0111001010;

    13'b01100100100z1 : data_tmp1 = 10'b0111001011;
 5  13'b0110010010z10 : data_tmp1 = 10'b0111001011;
    13'b0110010011z10z : data_tmp1 = 10'b0111001011;
    13'b0110010011z111 : data_tmp1 = 10'b0111001011;
    13'b0110010z110zz : data_tmp1 = 10'b0111001011;
    13'b0110010z11110 : data_tmp1 = 10'b0111001011;
10  13'b0110010010zzzz : data_tmp1 = 10'b0111001011;
    13'b0110010110zzzz : data_tmp1 = 10'b0111001011;
    13'b0110010111110z : data_tmp1 = 10'b0111001011;
    13'b0110010111111 : data_tmp1 = 10'b0111001011;
    13'b0110011000zzz : data_tmp1 = 10'b0111001011;
15  13'b01100110010000z : data_tmp1 = 10'b0111001011;
    13'b01100110010100 : data_tmp1 = 10'b0111001011;

    13'b0110011001z11 : data_tmp1 = 10'b0111001100;
    13'b0110011010z110z : data_tmp1 = 10'b0111001100;
20  13'b0110011010z1110 : data_tmp1 = 10'b0111001100;
    13'b0110011z10zzz : data_tmp1 = 10'b0111001100;
    13'b0110011z110zz : data_tmp1 = 10'b0111001100;
    13'b0110011z11111 : data_tmp1 = 10'b0111001100;
    13'b0110011110zzzz : data_tmp1 = 10'b0111001100;
25  13'b0110011111110z : data_tmp1 = 10'b0111001100;
    13'b01100111111110 : data_tmp1 = 10'b0111001100;
    13'b0110100000zz : data_tmp1 = 10'b0111001100;
    13'b01101000000100 : data_tmp1 = 10'b0111001100;

30  13'b0110100000z1z1 : data_tmp1 = 10'b0111001101;
    13'b011010000z110 : data_tmp1 = 10'b0111001101;
    13'b0110100000z10zz : data_tmp1 = 10'b0111001101;
    13'b0110100000z110z : data_tmp1 = 10'b0111001101;
    13'b0110100000z1111 : data_tmp1 = 10'b0111001101;
35  13'b0110100z10zzz : data_tmp1 = 10'b0111001101;
    13'b0110100z11110 : data_tmp1 = 10'b0111001101;
    13'b0110100010zzzz : data_tmp1 = 10'b0111001101;
    13'b0110100011110zz : data_tmp1 = 10'b0111001101;
    13'b01101000011110z : data_tmp1 = 10'b0111001101;
40  13'b01101000111111 : data_tmp1 = 10'b0111001101;

    13'b01101010zzzzz : data_tmp1 = 10'b0111001110;
    13'b011010110zzzz : data_tmp1 = 10'b0111001110;
    13'b01101011110zzz : data_tmp1 = 10'b0111001110;
45  13'b01101011110zz : data_tmp1 = 10'b0111001110;

    13'b01101011111zz : data_tmp1 = 10'b0111001111;
    13'b01101100zzzzz : data_tmp1 = 10'b0111001111;
    13'b0110110100zzzz : data_tmp1 = 10'b0111001111;
50  13'b011011010110zzz : data_tmp1 = 10'b0111001111;
    13'b01101101011000 : data_tmp1 = 10'b0111001111;

    13'b011011011110z1 : data_tmp1 = 10'b0111010000;
    13'b0110110111z10 : data_tmp1 = 10'b0111010000;
55  13'b0110110111110z : data_tmp1 = 10'b0111010000;
    13'b01101101111111 : data_tmp1 = 10'b0111010000;
```

```
      13'b01011010010zz : data_tmp1 = 10'b0111000100;
      13'b010110100110z : data_tmp1 = 10'b0111000100;
      13'b0101101001110 : data_tmp1 = 10'b0111000100;

 5    13'b01011010z1111 : data_tmp1 = 10'b0111000101;
      13'b0101101z10zzz : data_tmp1 = 10'b0111000101;
      13'b0101101z110zz : data_tmp1 = 10'b0111000101;
      13'b0101101z1110z : data_tmp1 = 10'b0111000101;
      13'b0101101z11110 : data_tmp1 = 10'b0111000101;
10    13'b010110110zzzz : data_tmp1 = 10'b0111000101;
      13'b0101101111111 : data_tmp1 = 10'b0111000101;
      13'b0101110000000z : data_tmp1 = 10'b0111000101;

      13'b0101110000z1z : data_tmp1 = 10'b0111000110;
15    13'b010111000z10z : data_tmp1 = 10'b0111000110;
      13'b01011100z10zz : data_tmp1 = 10'b0111000110;
      13'b01011100z111z : data_tmp1 = 10'b0111000110;
      13'b0101110010zzz : data_tmp1 = 10'b0111000110;
      13'b010111001110z : data_tmp1 = 10'b0111000110;
20    13'b010111010zzzz : data_tmp1 = 10'b0111000110;
      13'b01011101100zz : data_tmp1 = 10'b0111000110;
      13'b0101110110z : data_tmp1 = 10'b0111000110;
      13'b0101110110110 : data_tmp1 = 10'b0111000110;

25    13'b010111011z111 : data_tmp1 = 10'b0111000111;
      13'b010111101110zz : data_tmp1 = 10'b0111000111;
      13'b010111011110z : data_tmp1 = 10'b0111000111;
      13'b0101110111110 : data_tmp1 = 10'b0111000111;
      13'b01011110zzzzz : data_tmp1 = 10'b0111000111;
30    13'b0101111100zzz : data_tmp1 = 10'b0111000111;
      13'b01011111010zz : data_tmp1 = 10'b0111000111;

      13'b01011111z11zz : data_tmp1 = 10'b0111001000;
      13'b0101111110zzz : data_tmp1 = 10'b0111001000;
35    13'b010111111110zz : data_tmp1 = 10'b0111001000;
      13'b01100000zzzzz : data_tmp1 = 10'b0111001000;
      13'b0110000010000z : data_tmp1 = 10'b0111001000;

      13'b0110000100z1z : data_tmp1 = 10'b0111001001;
40    13'b011000010z10z : data_tmp1 = 10'b0111001001;
      13'b01100001z10zz : data_tmp1 = 10'b0111001001;
      13'b01100001z111z : data_tmp1 = 10'b0111001001;
      13'b0110000110zzz : data_tmp1 = 10'b0111001001;
      13'b011000011110z : data_tmp1 = 10'b0111001001;
45    13'b011000100zzzz : data_tmp1 = 10'b0111001001;
      13'b0110001010zzz : data_tmp1 = 10'b0111001001;
      13'b0110001011000 : data_tmp1 = 10'b0111001001;

      13'b01100010110z1 : data_tmp1 = 10'b0111001010;
50    13'b0110001011z10 : data_tmp1 = 10'b0111001010;
      13'b0110001z1110z : data_tmp1 = 10'b0111001010;
      13'b0110001z11111 : data_tmp1 = 10'b0111001010;
      13'b011000110zzzz : data_tmp1 = 10'b0111001010;
      13'b0110001110zzz : data_tmp1 = 10'b0111001010;
55    13'b01100011110zz : data_tmp1 = 10'b0111001010;
      13'b0110001111110 : data_tmp1 = 10'b0111001010;
```

132

```
        13'b01010000zzzzz : data_tmp1 = 10'b0110111110;
        13'b0101000100zzz : data_tmp1 = 10'b0110111110;
        13'b0101000101022 : data_tmp1 = 10'b0110111110;
        13'b0101000101100 : data_tmp1 = 10'b0110111110;
    5

        13'b01010001011z1 : data_tmp1 = 10'b0110111111;
        13'b01010001z1110 : data_tmp1 = 10'b0110111111;
        13'b0101000110zzz : data_tmp1 = 10'b0110111111;
        13'b0101000111022 : data_tmp1 = 10'b0110111111;
   10   13'b010100011110z : data_tmp1 = 10'b0110111111;
        13'b0101000111111 : data_tmp1 = 10'b0110111111;
        13'b0101001000zzzz : data_tmp1 = 10'b0110111111;
        13'b0101001010zzz : data_tmp1 = 10'b0110111111;
        13'b010100101100z : data_tmp1 = 10'b0110111111;
   15   13'b0101001011010 : data_tmp1 = 10'b0110111111;

        13'b0101001011z11 : data_tmp1 = 10'b0111000000;
        13'b0101001z1110z : data_tmp1 = 10'b0111000000;
        13'b0101001z11110 : data_tmp1 = 10'b0111000000;
   20   13'b010100110zzzz : data_tmp1 = 10'b0111000000;
        13'b0101001110zzz : data_tmp1 = 10'b0111000000;
        13'b01010011110zz : data_tmp1 = 10'b0111000000;
        13'b0101001111111 : data_tmp1 = 10'b0111000000;
        13'b0101010000zzz : data_tmp1 = 10'b0111000000;
   25   13'b010101000100z : data_tmp1 = 10'b0111000000;

        13'b0101010001z1z : data_tmp1 = 10'b0111000001;
        13'b01010100z110z : data_tmp1 = 10'b0111000001;
        13'b010101010z10zzz : data_tmp1 = 10'b0111000001;
   30   13'b010101000110zz : data_tmp1 = 10'b0111000001;
        13'b010101001111z : data_tmp1 = 10'b0111000001;
        13'b0101010010zzzz : data_tmp1 = 10'b0111000001;
        13'b0101010011100z : data_tmp1 = 10'b0111000001;

   35   13'b0101010111z1z : data_tmp1 = 10'b0111000010;
        13'b0101010011110z : data_tmp1 = 10'b0111000010;
        13'b01010110zzzzz : data_tmp1 = 10'b0111000010;
        13'b0101011100zzz : data_tmp1 = 10'b0111000010;
        13'b0101011110100z : data_tmp1 = 10'b0111000010;
   40   13'b0101011101010 : data_tmp1 = 10'b0111000010;

        13'b0101011101z11 : data_tmp1 = 10'b0111000011;
        13'b01010111z110z : data_tmp1 = 10'b0111000011;
        13'b01010111z1110 : data_tmp1 = 10'b0111000011;
   45   13'b0101011110zzz : data_tmp1 = 10'b0111000011;
        13'b0101010111110zz : data_tmp1 = 10'b0111000011;
        13'b0101011111111 : data_tmp1 = 10'b0111000011;
        13'b0101100000zzzz : data_tmp1 = 10'b0111000011;
        13'b0101100010zzz : data_tmp1 = 10'b0111000011;
   50   13'b01011000110zz : data_tmp1 = 10'b0111000011;

        13'b0101100z111zz : data_tmp1 = 10'b0111000100;
        13'b0101100010zzzz : data_tmp1 = 10'b0111000100;
        13'b0101100110zzz : data_tmp1 = 10'b0111000100;
   55   13'b0101100110zz : data_tmp1 = 10'b0111000100;
        13'b0101101000zzz : data_tmp1 = 10'b0111000100;
```

```
13':     000110100zz : data_tmp1 = 10'b0110110110;
13't    :0001101010z : data_tmp1 = 10'b0110110110;
13't   .100011010110 : data_tmp1 = 10'b0110110110;

5        13't .10001101z111 : data_tmp1 = 10'b0110110111;
         13'b0100011z110zz : data_tmp1 = 10'b0110110111;
         13'b0100011z1110z : data_tmp1 = 10'b0110110111;
         13'b0100011z11110 : data_tmp1 = 10'b0110110111;
         13'b010001110zzzz : data_tmp1 = 10'b0110110111;
10       13'b0100011110zzz : data_tmp1 = 10'b0110110111;

         13'b0100011111111 : data_tmp1 = 10'b0110111000;
         13'b01001000zzzzz : data_tmp1 = 10'b0110111000;
         13'b0100100100zzz : data_tmp1 = 10'b0110111000;
15
         13'b01001001z1zzz : data_tmp1 = 10'b0110111001;
         13'b0100100110zzz : data_tmp1 = 10'b0110111001;
         13'b010010100zzzz : data_tmp1 = 10'b0110111001;
         13'b0100101010000 : data_tmp1 = 10'b0110111001;
20
         13'b01001010100z1 : data_tmp1 = 10'b0110111010;
         13'b0100101010z10 : data_tmp1 = 10'b0110111010;
         13'b010010101z10z : data_tmp1 = 10'b0110111010;
         13'b010010101z111 : data_tmp1 = 10'b0110111010;
25       13'b0100101z110zz : data_tmp1 = 10'b0110111010;
         13'b0100101011110 : data_tmp1 = 10'b0110111010;
         13'b010010110zzzz : data_tmp1 = 10'b0110111010;
         13'b0100101110zzz : data_tmp1 = 10'b0110111010;

30       13'b01001011111zz : data_tmp1 = 10'b0110111011;
         13'b01001100zzzzz : data_tmp1 = 10'b0110111011;
         13'b01001101000zz : data_tmp1 = 10'b0110111011;
         13'b010011010010z : data_tmp1 = 10'b0110111011;
         13'b0100110100110 : data_tmp1 = 10'b0110111011;
35
         13'b010011010z111 : data_tmp1 = 10'b0110111100;
         13'b01001101z10zz : data_tmp1 = 10'b0110111100;
         13'b01001101z110z : data_tmp1 = 10'b0110111100;
         13'b01001101z1110 : data_tmp1 = 10'b0110111100;
40       13'b0100110110zzz : data_tmp1 = 10'b0110111100;
         13'b0100110111111 : data_tmp1 = 10'b0110111100;
         13'b010011100zzzz : data_tmp1 = 10'b0110111100;
         13'b010011101000z : data_tmp1 = 10'b0110111100;
         13'b0100111010010 : data_tmp1 = 10'b0110111100;
45
         13'b0100111010z11 : data_tmp1 = 10'b0110111101;
         13'b010011101z10z : data_tmp1 = 10'b0110111101;
         13'b010011101z110 : data_tmp1 = 10'b0110111101;
         13'b0100111z110zz : data_tmp1 = 10'b0110111101;
50       13'b0100111011111 : data_tmp1 = 10'b0110111101;
         13'b010011110zzzz : data_tmp1 = 10'b0110111101;
         13'b0100111110zzz : data_tmp1 = 10'b0110111101;
         13'b010011111110z : data_tmp1 = 10'b0110111101;
         13'b0100111111110 : data_tmp1 = 10'b0110111101;
55
         13'b0100111111111 : data_tmp1 = 10'b0110111110;
```

```
        13'b0011110010zzz : data_tmp1 = 10'b0110101110;
        13'b00111100110zz : data_tmp1 = 10'b0110101110;
        13'b0011110011111 : data_tmp1 = 10'b0110101110;
        13'b0011110100zzz : data_tmp1 = 10'b0110101110;
    5   13'b00111101010zz : data_tmp1 = 10'b0110101110;
        13'b0011110101100 : data_tmp1 = 10'b0110101110;

        13'b00111101011z1 : data_tmp1 = 10'b0110101111;
        13'b00111101z1110 : data_tmp1 = 10'b0110101111;
   10   13'b0011110110zzz : data_tmp1 = 10'b0110101111;
        13'b00111101110zz : data_tmp1 = 10'b0110101111;
        13'b0011110111110z : data_tmp1 = 10'b0110101111;
        13'b0011110111111 : data_tmp1 = 10'b0110101111;
        13'b001111100zzzz : data_tmp1 = 10'b0110101111;
   15
        13'b001111101zzzz : data_tmp1 = 10'b0110110000;
        13'b001111110zzzz : data_tmp1 = 10'b0110110000;
        13'b00111111100zz : data_tmp1 = 10'b0110110000;

   20   13'b00111111111z1zz : data_tmp1 = 10'b0110110001;
        13'b00111111110zz : data_tmp1 = 10'b0110110001;
        13'b010000000zzzz : data_tmp1 = 10'b0110110001;
        13'b0100000010zzz : data_tmp1 = 10'b0110110001;

   25   13'b0100000011zzz : data_tmp1 = 10'b0110110010;
        13'b010000010zzzz : data_tmp1 = 10'b0110110010;
        13'b0100000110zzz : data_tmp1 = 10'b0110110010;
        13'b01000001110zz : data_tmp1 = 10'b0110110010;
        13'b0100000111100 : data_tmp1 = 10'b0110110010;
   30
        13'b01000001111z1 : data_tmp1 = 10'b0110110011;
        13'b0100000111110 : data_tmp1 = 10'b0110110011;
        13'b01000010zzzzz : data_tmp1 = 10'b0110110011;
        13'b010000110000z : data_tmp1 = 10'b0110110011;
   35
        13'b0100001100z1z : data_tmp1 = 10'b0110110100;
        13'b010000110z10z : data_tmp1 = 10'b0110110100;
        13'b01000011z10zz : data_tmp1 = 10'b0110110100;
        13'b01000011z111z : data_tmp1 = 10'b0110110100;
   40   13'b0100001110zzz : data_tmp1 = 10'b0110110100;
        13'b010000111110z : data_tmp1 = 10'b0110110100;
        13'b0100010000zzz : data_tmp1 = 10'b0110110100;

        13'b01000100z1zzz : data_tmp1 = 10'b0110110101;
   45   13'b0100010010zzz : data_tmp1 = 10'b0110110101;
        13'b0100010100zzz : data_tmp1 = 10'b0110110101;
        13'b010001010100zz : data_tmp1 = 10'b0110110101;
        13'b010001010110z : data_tmp1 = 10'b0110110101;
        13'b0100010101110 : data_tmp1 = 10'b0110110101;
   50
        13'b01000101z1111 : data_tmp1 = 10'b0110110110;
        13'b0100010110zzz : data_tmp1 = 10'b0110110110;
        13'b01000101110zz : data_tmp1 = 10'b0110110110;
        13'b010001011110z : data_tmp1 = 10'b0110110110;
   55   13'b0100010111110 : data_tmp1 = 10'b0110110110;
        13'b010001100zzzz : data_tmp1 = 10'b0110110110;
```

```
        13'b0011101010100z : data_tmp1 = 10'b0110100110;
        13'b0011010101010 : data_tmp1 = 10'b0110100110;

        13'b0011010101z11 : data_tmp1 = 10'b0110100111;
    5   13'b00110101z110z : data_tmp1 = 10'b0110100111;
        13'b00110101z1110 : data_tmp1 = 10'b0110100111;
        13'b0011010110zzz : data_tmp1 = 10'b0110100111;
        13'b00110101110zz : data_tmp1 = 10'b0110100111;
        13'b0011010111111 : data_tmp1 = 10'b0110100111;
   10   13'b0011011000zzz : data_tmp1 = 10'b0110100111;
        13'b001101100100z : data_tmp1 = 10'b0110100111;

        13'b0011011001z1z : data_tmp1 = 10'b0110101000;
        13'b00110110z110z : data_tmp1 = 10'b0110101000;
   15   13'b0011011010zzz : data_tmp1 = 10'b0110101000;
        13'b00110110110zz : data_tmp1 = 10'b0110101000;
        13'b001101101111z : data_tmp1 = 10'b0110101000;
        13'b0011011100zzz : data_tmp1 = 10'b0110101000;
        13'b0011011101000 : data_tmp1 = 10'b0110101000;
   20
        13'b001101110z1 : data_tmp1 = 10'b0110101001;
        13'b0011011101z10 : data_tmp1 = 10'b0110101001;
        13'b00110111z110z : data_tmp1 = 10'b0110101001;
        13'b00110111z1111 : data_tmp1 = 10'b0110101001;
   25   13'b00110111110zzz : data_tmp1 = 10'b0110101001;
        13'b00110111110zz : data_tmp1 = 10'b0110101001;
        13'b0011011111110 : data_tmp1 = 10'b0110101001;
        13'b0011100000zzz : data_tmp1 = 10'b0110101001;

   30   13'b00111000z1zzz : data_tmp1 = 10'b0110101010;
        13'b0011100010zzz : data_tmp1 = 10'b0110101010;
        13'b0011100100zzz : data_tmp1 = 10'b0110101010;

        13'b00111001z1zzz : data_tmp1 = 10'b0110101011;
   35   13'b0011100110zzz : data_tmp1 = 10'b0110101011;
        13'b0011101000zzz : data_tmp1 = 10'b0110101011;

        13'b00111010z1zzz : data_tmp1 = 10'b0110101100;
        13'b0011101010zzz : data_tmp1 = 10'b0110101100;
   40   13'b0011101100zzz : data_tmp1 = 10'b0110101100;
        13'b0011101101000 : data_tmp1 = 10'b0110101100;

        13'b00111011010z1 : data_tmp1 = 10'b0110101101;
        13'b0011101101z10 : data_tmp1 = 10'b0110101101;
   45   13'b00111011z110z : data_tmp1 = 10'b0110101101;
        13'b00111011z1111 : data_tmp1 = 10'b0110101101;
        13'b0011101110zzz : data_tmp1 = 10'b0110101101;
        13'b00111011110zz : data_tmp1 = 10'b0110101101;
        13'b0011101111110 : data_tmp1 = 10'b0110101101;
   50   13'b0011110000zzz : data_tmp1 = 10'b0110101101;
        13'b001111000100z : data_tmp1 = 10'b0110101101;
        13'b0011110001010 : data_tmp1 = 10'b0110101101;

        13'b0011110001z11 : data_tmp1 = 10'b0110101110;
   55   13'b00111100z110z : data_tmp1 = 10'b0110101110;
        13'b00111100z1110 : data_tmp1 = 10'b0110101110;
```

```
        13'b0010011011z10z : data_tmp1 = 10'b0110011110;
        13'b0010011011z111 : data_tmp1 = 10'b0110011110;
        13'b00101101110zz : data_tmp1 = 10'b0110011110;
        13'b00010110111110 : data_tmp1 = 10'b0110011110;
    5   13'b00010111000zzz : data_tmp1 = 10'b0110011110;
        13'b0010011100100z : data_tmp1 = 10'b0110011110;
        13'b0010011100101010 : data_tmp1 = 10'b0110011110;

        13'b00010111001z11 : data_tmp1 = 10'b0110011111;
   10   13'b00101110z110z : data_tmp1 = 10'b0110011111;
        13'b00101110z1110 : data_tmp1 = 10'b0110011111;
        13'b00010111010zzz : data_tmp1 = 10'b0110011111;
        13'b0010011110110zz : data_tmp1 = 10'b0110011111;
        13'b00010111011111 : data_tmp1 = 10'b0110011111;
        13'b00101111000zz : data_tmp1 = 10'b0110011111;
   15   13'b0010111100100 : data_tmp1 = 10'b0110011111;

        13'b00101111001z1 : data_tmp1 = 10'b0110100000;
        13'b0010011110z110 : data_tmp1 = 10'b0110100000;
        13'b00101111z10zz : data_tmp1 = 10'b0110100000;
   20   13'b00101111z110z : data_tmp1 = 10'b0110100000;
        13'b00101111z1111 : data_tmp1 = 10'b0110100000;
        13'b00010111110zzz : data_tmp1 = 10'b0110100000;
        13'b00010111111110 : data_tmp1 = 10'b0110100000;
   25
        13'b001100000zzzz : data_tmp1 = 10'b0110100001;
        13'b0011000010zzz : data_tmp1 = 10'b0110100001;
        13'b0011000000110zz : data_tmp1 = 10'b0110100001;

   30   13'b00110000111zz : data_tmp1 = 10'b0110100010;
        13'b001100010zzzz : data_tmp1 = 10'b0110100010;
        13'b00110001100zz : data_tmp1 = 10'b0110100010;
        13'b001100011010z : data_tmp1 = 10'b0110100010;
        13'b0011000110110 : data_tmp1 = 10'b0110100010;
   35
        13'b001100011z111 : data_tmp1 = 10'b0110100011;
        13'b00110001110zz : data_tmp1 = 10'b0110100011;
        13'b001100011110z : data_tmp1 = 10'b0110100011;
        13'b0011000111110 : data_tmp1 = 10'b0110100011;
   40   13'b001100100zzzz : data_tmp1 = 10'b0110100011;
        13'b00110010100zz : data_tmp1 = 10'b0110100011;

        13'b001100101z1zz : data_tmp1 = 10'b0110100100;
        13'b00110010110zz : data_tmp1 = 10'b0110100100;
   45   13'b001100110zzzz : data_tmp1 = 10'b0110100100;

        13'b001100111zzzz : data_tmp1 = 10'b0110100101;
        13'b0011010000zzz : data_tmp1 = 10'b0110100101;
        13'b00110100010zz : data_tmp1 = 10'b0110100101;
   50   13'b001101000110z : data_tmp1 = 10'b0110100101;

        13'b00110100z111z : data_tmp1 = 10'b0110100110;
        13'b0011010010zzz : data_tmp1 = 10'b0110100110;
        13'b00110100110zz : data_tmp1 = 10'b0110100110;
   55   13'b001101001110z : data_tmp1 = 10'b0110100110;
        13'b0011010100zzz : data_tmp1 = 10'b0110100110;
```

```
13'b001001100z11z : data_tmp1 = 10'b0110010100;
13'b00100110z10zz : data_tmp1 = 10'b0110010100;
13'b001001100110z : data_tmp1 = 10'b0110010100;
13'b0010011010zzz : data_tmp1 = 10'b0110010100;

13'b001001110111zz : data_tmp1 = 10'b0110010101;
13'b001001110zzzz : data_tmp1 = 10'b0110010101;
13'b001001111000z : data_tmp1 = 10'b0110010101;

13'b0010011110z1z : data_tmp1 = 10'b0110010110;
13'b001001111z10z : data_tmp1 = 10'b0110010110;
13'b00100111110zz : data_tmp1 = 10'b0110010110;
13'b0010011111111z : data_tmp1 = 10'b0110010110;
13'b0010100000zzz : data_tmp1 = 10'b0110010110;

13'b0010100001zzz : data_tmp1 = 10'b0110010111;
13'b0010100010zzz : data_tmp1 = 10'b0110010111;
13'b00101000110zz : data_tmp1 = 10'b0110010111;
13'b001010001110z : data_tmp1 = 10'b0110010111;
13'b0010100011110 : data_tmp1 = 10'b0110010111;

13'b0010100011111 : data_tmp1 = 10'b0110011000;
13'b001010010zzzz : data_tmp1 = 10'b0110011000;
13'b00101001100zz : data_tmp1 = 10'b0110011000;
13'b00101001100zz : data_tmp1 = 10'b0110011000;
13'b00101001101 0z : data_tmp1 = 10'b0110011000;

13'b001010011z11z : data_tmp1 = 10'b0110011001;
13'b00101001110zz : data_tmp1 = 10'b0110011001;
13'b00101001111 0z : data_tmp1 = 10'b0110011001;
13'b0010101000zzz : data_tmp1 = 10'b0110011001;
13'b00101010010zz : data_tmp1 = 10'b0110011001;
13'b001010100110z : data_tmp1 = 10'b0110011001;

13'b00101010z111z : data_tmp1 = 10'b0110011010;
13'b0010101010zzz : data_tmp1 = 10'b0110011010;
13'b0010101010110zz : data_tmp1 = 10'b0110011010;
13'b0010101010110z : data_tmp1 = 10'b0110011010;
13'b00101011000zz : data_tmp1 = 10'b0110011010;
13'b001010110010z : data_tmp1 = 10'b0110011010;

13'b001010110z11z : data_tmp1 = 10'b0110011011;
13'b00101011z10zz : data_tmp1 = 10'b0110011011;
13'b00101011z110z : data_tmp1 = 10'b0110011011;
13'b0010101110zzz : data_tmp1 = 10'b0110011011;
13'b0010101111110 : data_tmp1 = 10'b0110011011;

13'b0010101111111 : data_tmp1 = 10'b0110011100;
13'b001011000zzzz : data_tmp1 = 10'b0110011100;
13'b0010110010zzz : data_tmp1 = 10'b0110011100;

13'b0010110011zzz : data_tmp1 = 10'b0110011101;
13'b001011010zzzz : data_tmp1 = 10'b0110011101;
13'b0010110110000 : data_tmp1 = 10'b0110011101;

13'b00101101100z1 : data_tmp1 = 10'b0110011110;
13'b0010110110z10 : data_tmp1 = 10'b0110011110;
```

```
      13'b001    )010z11 : data_tmp1 = 10'b0110001011;
      13'b001    )01z10z : data_tmp1 = 10'b0110001011;
      13'b001C..001z110 : data_tmp1 = 10'b0110001011;
      13'b00100000110zz : data_tmp1 = 10'b0110001011;
 5    13'b00100000011111 : data_tmp1 = 10'b0110001011;
      13'b00100001000zz : data_tmp1 = 10'b0110001011;
      13'b001000010010z : data_tmp1 = 10'b0110001011;

      13'b001000010z11z : data_tmp1 = 10'b0110001100;
10    13'b00100001010zz : data_tmp1 = 10'b0110001100;
      13'b0010000;0110z : data_tmp1 = 10'b0110001100;
      13'b0010000110zzz : data_tmp1 = 10'b0110001100;

      13'b0010000111zzz : data_tmp1 = 10'b0110001101;
15    13'b0010001000zzz : data_tmp1 = 10'b0110001101;
      13'b00100010010zz : data_tmp1 = 10'b0110001101;

      13'b00100010011zz : data_tmp1 = 10'b0110001110;
      13'b0010001010zzz : data_tmp1 = 10'b0110001110;
20    13'b00100010110zz : data_tmp1 = 10'b0110001110;
      13'b0010000101110z : data_tmp1 = 10'b0110001110;
      13'b00100001011110 : data_tmp1 = 10'b0110001110;

      13'b0010001011111 : data_tmp1 = 10'b0110001111;
25    13'b001000110zzzz : data_tmp1 = 10'b0110001111;
      13'b001000111000z : data_tmp1 = 10'b0110001111;
      13'b0010001110010 : data_tmp1 = 10'b0110001111;

      13'b0010001110z11 : data_tmp1 = 10'b0110010000;
30    13'b001000111z10z : data_tmp1 = 10'b0110010000;
      13'b001000111z110 : data_tmp1 = 10'b0110010000;
      13'b00100011110zz : data_tmp1 = 10'b0110010000;
      13'b0010001111111 : data_tmp1 = 10'b0110010000;
      13'b001001000000zz : data_tmp1 = 10'b0110010000;
35    13'b001001000010z : data_tmp1 = 10'b0110010000;
      13'b0010010000110 : data_tmp1 = 10'b0110010000;

      13'b001001000z111 : data_tmp1 = 10'b0110010001;
      13'b00100100z10zz : data_tmp1 = 10'b0110010001;
40    13'b001001000110z : data_tmp1 = 10'b0110010001;
      13'b00100100001110 : data_tmp1 = 10'b0110010001;
      13'b00100100010zzz : data_tmp1 = 10'b0110010001;

      13'b00100100111zz : data_tmp1 = 10'b0110010010;
45    13'b001001010zzzz : data_tmp1 = 10'b0110010010;
      13'b00100010110000 : data_tmp1 = 10'b0110010010;

      13'b001001011100z1 : data_tmp1 = 10'b0110010011;
      13'b0010010110z10 : data_tmp1 = 10'b0110010011;
50    13'b001001011z10z : data_tmp1 = 10'b0110010011;
      13'b001001011z111 : data_tmp1 = 10'b0110010011;
      13'b00100101110zz : data_tmp1 = 10'b0110010011;
      13'b00100010111110 : data_tmp1 = 10'b0110010011;
      13'b00100110000zz : data_tmp1 = 10'b0110010011;
55    13'b001001100010z : data_tmp1 = 10'b0110010011;
```

125

```
        13'b 001101111z1z : data_tmp1 = 10'b0110000010;
        13'b 00110111110z : data_tmp1 = 10'b0110000010;
        13'b0001110000zzz : data_tmp1 = 10'b0110000010;
        13'b000111000100z : data_tmp1 = 10'b0110000010;
 5
        13'b0001110001z1z : data_tmp1 = 10'b0110000011;
        13'b000111000110z : data_tmp1 = 10'b0110000011;
        13'b0001110010zzz : data_tmp1 = 10'b0110000011;
        13'b000111001100z : data_tmp1 = 10'b0110000011;
10
        13'b0001110011z1z : data_tmp1 = 10'b0110000100;
        13'b000111001110z : data_tmp1 = 10'b0110000100;
        13'b0001110100zzz : data_tmp1 = 10'b0110000100;
        13'b000111010100z : data_tmp1 = 10'b0110000100;
15      13'b0001110101010 : data_tmp1 = 10'b0110000100;

        13'b0001110101z11 : data_tmp1 = 10'b0110000101;
        13'b000111010110z : data_tmp1 = 10'b0110000101;
        13'b0001110101110 : data_tmp1 = 10'b0110000101;
20      13'b0001110110zzz : data_tmp1 = 10'b0110000101;
        13'b000111011100z : data_tmp1 = 10'b0110000101;
        13'b0001110111010 : data_tmp1 = 10'b0110000101;

        13'b0001110111z11 : data_tmp1 = 10'b0110000110;
25      13'b000111011110z : data_tmp1 = 10'b0110000110;
        13'b0001110111110 : data_tmp1 = 10'b0110000110;
        13'b0001111000zzz : data_tmp1 = 10'b0110000110;
        13'b000111110010zz : data_tmp1 = 10'b0110000110;

30      13'b00011110011zz : data_tmp1 = 10'b0110000111;
        13'b0001111010zzz : data_tmp1 = 10'b0110000111;
        13'b00011110110zz : data_tmp1 = 10'b0110000111;
        13'b0001111011100 : data_tmp1 = 10'b0110000111;

35      13'b00011110111z1 : data_tmp1 = 10'b0110001000;
        13'b0001111011110 : data_tmp1 = 10'b0110001000;
        13'b0001111100zzz : data_tmp1 = 10'b0110001000;
        13'b00011111010zz : data_tmp1 = 10'b0110001000;
        13'b000111110110z : data_tmp1 = 10'b0110001000;
40      13'b0001111101110 : data_tmp1 = 10'b0110001000;

        13'b0001111z1111 : data_tmp1 = 10'b0110001001;
        13'b0001111110zzz : data_tmp1 = 10'b0110001001;
        13'b00011111110zz : data_tmp1 = 10'b0110001001;
45      13'b0001111111110z : data_tmp1 = 10'b0110001001;
        13'b0001111111110 : data_tmp1 = 10'b0110001001;
        13'b0010000000000 : data_tmp1 = 10'b0110001001;

        13'b00100000000z1 : data_tmp1 = 10'b0110001010;
50      13'b0010000000z10 : data_tmp1 = 10'b0110001010;
        13'b001000000z10z : data_tmp1 = 10'b0110001010;
        13'b001000000z111 : data_tmp1 = 10'b0110001010;
        13'b00100000010zz : data_tmp1 = 10'b0110001010;
        13'b0010000001110 : data_tmp1 = 10'b0110001010;
55      13'b0010000001000z : data_tmp1 = 10'b0110001010;
        13'b0010000010010 : data_tmp1 = 10'b0110001010;
```

```
      13'b000101110110z : data_tmp1 = 10'b0101111000;
      13'b00010111101110 : data_tmp1 = 10'b0101111000;
      13'b0010011110zzz : data_tmp1 = 10'b0101111000;

 5    13'b00010111111zzz : data_tmp1 = 10'b0101111001;
      13'b00011000000zz : data_tmp1 = 10'b0101111001;
      13'b0001100000100 : data_tmp1 = 10'b0101111001;

      13'b00011000001z1 : data_tmp1 = 10'b0101111010;
10    13'b000110000z110 : data_tmp1 = 10'b0101111010;
      13'b00011000010zz : data_tmp1 = 10'b0101111010;
      13'b000110000110z : data_tmp1 = 10'b0101111010;
      13'b0001100001111 : data_tmp1 = 10'b0101111010;
      13'b000110001000z : data_tmp1 = 10'b0101111010;
15    13'b0001100010010 : data_tmp1 = 10'b0101111010;

      13'b0001100010z11 : data_tmp1 = 10'b0101111011;
      13'b000110001z10z : data_tmp1 = 10'b0101111011;
      13'b000110001z110 : data_tmp1 = 10'b0101111011;
20    13'b00011000110zz : data_tmp1 = 10'b0101111011;
      13'b0001100011111 : data_tmp1 = 10'b0101111011;
      13'b0001100100000 : data_tmp1 = 10'b0101111011;

      13'b00011001000z1 : data_tmp1 = 10'b0101111100;
25    13'b0001100100z10 : data_tmp1 = 10'b0101111100;
      13'b000110010z10z : data_tmp1 = 10'b0101111100;
      13'b0001100100111 : data_tmp1 = 10'b0101111100;
      13'b00011001010zz : data_tmp1 = 10'b0101111100;
      13'b0001100101110 : data_tmp1 = 10'b0101111100;
30
      13'b0001100101111 : data_tmp1 = 10'b0101111101;
      13'b0001100110zzz : data_tmp1 = 10'b0101111101;
      13'b00011001110zz : data_tmp1 = 10'b0101111101;
      13'b000110011110z : data_tmp1 = 10'b0101111101;
35
      13'b0001100111111z : data_tmp1 = 10'b0101111110;
      13'b0001101000zzz : data_tmp1 = 10'b0101111110;
      13'b00011010010zz : data_tmp1 = 10'b0101111110;
      13'b0001101001100 : data_tmp1 = 10'b0101111110;
40
      13'b00011010011z1 : data_tmp1 = 10'b0101111111;
      13'b0001101001110 : data_tmp1 = 10'b0101111111;
      13'b0001101010zzz : data_tmp1 = 10'b0101111111;
      13'b000110010110zz : data_tmp1 = 10'b0101111111;
45
      13'b000110010111zz : data_tmp1 = 10'b0110000000;
      13'b0001101100zzz : data_tmp1 = 10'b0110000000;
      13'b000110110100z : data_tmp1 = 10'b0110000000;
      13'b0001101101010 : data_tmp1 = 10'b0110000000;
50
      13'b0001101101z11 : data_tmp1 = 10'b0110000001;
      13'b000110110110z : data_tmp1 = 10'b0110000001;
      13'b0001101101110 : data_tmp1 = 10'b0110000001;
      13'b0001101110zzz : data_tmp1 = 10'b0110000001;
55    13'b000110111100z : data_tmp1 = 10'b0110000001;
```

```
13'b00010010111z1 : data_tmp1 = 10'b0101101100;
13'b00010010111110 : data_tmp1 = 10'b0101101100;
13'b000100011000zz : data_tmp1 = 10'b0101101100;
13'b000100110010z : data_tmp1 = 10'b0101101100;
13'b0001001100110 : data_tmp1 = 10'b0101101100;

13'b000100110z111 : data_tmp1 = 10'b0101101101;
13'b00010011010zz : data_tmp1 = 10'b0101101101;
13'b000100110110z : data_tmp1 = 10'b0101101101;
13'b0001001101110 : data_tmp1 = 10'b0101101101;
13'b000100111000z : data_tmp1 = 10'b0101101101;

13'b0001001110z1z : data_tmp1 = 10'b0101101110;
13'b000100111010z : data_tmp1 = 10'b0101101110;
13'b00010011110zz : data_tmp1 = 10'b0101101110;
13'b0001001111100 : data_tmp1 = 10'b0101101110;

13'b00010011111z1 : data_tmp1 = 10'b0101101111;
13'b0001001111110 : data_tmp1 = 10'b0101101111;
13'b0001010000zzz : data_tmp1 = 10'b0101101111;
13'b0001010001000 : data_tmp1 = 10'b0101101111;

13'b00010100010z1 : data_tmp1 = 10'b0101110000;
13'b0001010001z10 : data_tmp1 = 10'b0101110000;
13'b000101000110z : data_tmp1 = 10'b0101110000;
13'b0001010001111 : data_tmp1 = 10'b0101110000;
13'b00010100100zz : data_tmp1 = 10'b0101110000;

13'b000101001z1zz : data_tmp1 = 10'b0101110001;
13'b00010100110zz : data_tmp1 = 10'b0101110001;

13'b0001010100zzz : data_tmp1 = 10'b0101110010;
13'b00010101010zz : data_tmp1 = 10'b0101110010;

13'b00010101011zz : data_tmp1 = 10'b0101110011;
13'b0001010110zzz : data_tmp1 = 10'b0101110011;

13'b00010101011zzz : data_tmp1 = 10'b0101110100;
13'b000101100000zz : data_tmp1 = 10'b0101110100;

13'b000101100z1zz : data_tmp1 = 10'b0101110101;
13'b00010110010zz : data_tmp1 = 10'b0101110101;
13'b0001011010000 : data_tmp1 = 10'b0101110101;

13'b00010110100z1 : data_tmp1 = 10'b0101110110;
13'b0001011010z10 : data_tmp1 = 10'b0101110110;
13'b000101101z10z : data_tmp1 = 10'b0101110110;
13'b0001011010111 : data_tmp1 = 10'b0101110110;
13'b00010110110zz : data_tmp1 = 10'b0101110110;

13'b000101101111z : data_tmp1 = 10'b0101110111;
13'b0001011100zzz : data_tmp1 = 10'b0101110111;
13'b000101110100z : data_tmp1 = 10'b0101110111;
13'b0001011101010 : data_tmp1 = 10'b0101110111;

13'b0001011101z11 : data_tmp1 = 10'b0101111000;
```

```
      13'b00000111100z10 : data_tmp1 = 10'b0101011111;
      13'b0000011110010z : data_tmp1 = 10'b0101011111;
      13'b00000111100111 : data_tmp1 = 10'b0101011111;
      13'b000011110100z : data_tmp1 = 10'b0101011111;
  5
      13'b00000111101z1z : data_tmp1 = 10'b0101100000;
      13'b000011110110z : data_tmp1 = 10'b0101100000;
      13'b0000011111000z : data_tmp1 = 10'b0101100000;

 10   13'b0000011110z1z : data_tmp1 = 10'b0101100001;
      13'b000011111010z : data_tmp1 = 10'b0101100001;
      13'b000011111100z : data_tmp1 = 10'b0101100001;
      13'b00000111111010 : data_tmp1 = 10'b0101100001;

 15   13'b00000111111z11 : data_tmp1 = 10'b0101100010;
      13'b000011111110z : data_tmp1 = 10'b0101100010;
      13'b0000011111110 : data_tmp1 = 10'b0101100010;
      13'b00010000000zz : data_tmp1 = 10'b0101100010;

 20   13'b00010000001zz : data_tmp1 = 10'b0101100011;
      13'b00010000010zz : data_tmp1 = 10'b0101100011;
      13'b0001000001100 : data_tmp1 = 10'b0101100011;

      13'b00010000011z1 : data_tmp1 = 10'b0101100100;
 25   13'b0001000001110 : data_tmp1 = 10'b0101100100;
      13'b00010000100zz : data_tmp1 = 10'b0101100100;
      13'b000100001010z : data_tmp1 = 10'b0101100100;
      13'b0001000010110 : data_tmp1 = 10'b0101100100;

 30   13'b000100001z111 : data_tmp1 = 10'b0101100101;
      13'b00010000110zz : data_tmp1 = 10'b0101100101;
      13'b000100001110z : data_tmp1 = 10'b0101100101;
      13'b0001000011110 : data_tmp1 = 10'b0101100101;

 35   13'b0001000100zzz : data_tmp1 = 10'b0101100110;
      13'b000100010100z : data_tmp1 = 10'b0101100110;

      13'b0001000101z1z : data_tmp1 = 10'b0101100111;
      13'b000100010110z : data_tmp1 = 10'b0101100111;
 40   13'b00010001100zz : data_tmp1 = 10'b0101100111;

      13'b00010001101zz : data_tmp1 = 10'b0101101000;
      13'b00010001110zz : data_tmp1 = 10'b0101101000;
      13'b000100011110z : data_tmp1 = 10'b0101101000;
 45
      13'b000100011111z : data_tmp1 = 10'b0101101001;
      13'b0001001000zzz : data_tmp1 = 10'b0101101001;

      13'b0001001001zzz : data_tmp1 = 10'b0101101010;
 50   13'b000100101000z : data_tmp1 = 10'b0101101010;

      13'b0001001010z1z : data_tmp1 = 10'b0101101011;
      13'b000100101010z : data_tmp1 = 10'b0101101011;
      13'b00010010110zz : data_tmp1 = 10'b0101101011;
 55   13'b0001001011100 : data_tmp1 = 10'b0101101011;
```

```
13'b00001100000zz : data_tmp1 = 10'b0101010010;
13'b000011000010z : data_tmp1 = 10'b0101010010;

13'b0000011000011z : data_tmp1 = 10'b0101010011;
13'b00001100010zz : data_tmp1 = 10'b0101010011;

13'b00001100011zz : data_tmp1 = 10'b0101010100;
13'b000011001000z : data_tmp1 = 10'b0101010100;
13'b00000110010010 : data_tmp1 = 10'b0101010100;

13'b0000110010z11 : data_tmp1 = 10'b0101010101;
13'b000011001010z : data_tmp1 = 10'b0101010101;
13'c00000110010110 : data_tmp1 = 10'b0101010101;
13'b000011001100z : data_tmp1 = 10'b0101010101;
13'b00000110011010 : data_tmp1 = 10'b0101010101;

13'b0000110011z11 : data_tmp1 = 10'b0101010110;
13'b000011001110z : data_tmp1 = 10'b0101010110;
13'b00000110011110 : data_tmp1 = 10'b0101010110;
13'b000011010000z : data_tmp1 = 10'b0101010110;

13'b0000110100z1z : data_tmp1 = 10'b0101010111;
13'b000011010010z : data_tmp1 = 10'b0101010111;
13'b00000110101000 : data_tmp1 = 10'b0101010111;

13'b00001101010z1 : data_tmp1 = 10'b0101011000;
13'b0000110101z10 : data_tmp1 = 10'b0101011000;
13'b000011010110z : data_tmp1 = 10'b0101011000;
13'b00000110101111 : data_tmp1 = 10'b0101011000;
13'b00000110110000 : data_tmp1 = 10'b0101011000;

13'b00001101100z1 : data_tmp1 = 10'b0101011001;
13'b0000110110z10 : data_tmp1 = 10'b0101011001;
13'b000011011010z : data_tmp1 = 10'b0101011001;
13'b00000110110111 : data_tmp1 = 10'b0101011001;
13'b00000110111000 : data_tmp1 = 10'b0101011001;

13'b00001101110z1 : data_tmp1 = 10'b0101011010;
13'b0000110111z10 : data_tmp1 = 10'b0101011010;
13'b000011011110z : data_tmp1 = 10'b0101011010;
13'b00000110111111 : data_tmp1 = 10'b0101011010;

13'b0000111000zzz : data_tmp1 = 10'b0101011011;

13'b0000111001zzz : data_tmp1 = 10'b0101011100;

13'b0000111010zzz : data_tmp1 = 10'b0101011101;
13'b00000111011000 : data_tmp1 = 10'b0101011101;

13'b00001110110z1 : data_tmp1 = 10'b0101011110;
13'b0000111011z10 : data_tmp1 = 10'b0101011110;
13'b000011101110z : data_tmp1 = 10'b0101011110;
13'b00000111011111 : data_tmp1 = 10'b0101011110;
13'b00000111100000 : data_tmp1 = 10'b0101011110;

13'b000001111000z1 : data_tmp1 = 10'b0101011111;
```

```
       13'b00001001100z1 : data_tmp1 = 10'b0101000101;
       13'b0000100110010 : data_tmp1 = 10'b0101000101;
       13'b000010011010z : data_tmp1 = 10'b0101000101;

  5    13'b000010011011z : data_tmp1 = 10'b0101000110;
       13'b000010011100z : data_tmp1 = 10'b0101000110;
       13'b0000100111010 : data_tmp1 = 10'b0101000110;

       13'b00001001111z11 : data_tmp1 = 10'b0101000111;
 10    13'b000010011110z : data_tmp1 = 10'b0101000111;
       13'b0000100111110 : data_tmp1 = 10'b0101000111;
       13'b0000101000000 : data_tmp1 = 10'b0101000111;

       13'b00001010000z1 : data_tmp1 = 10'b0101001000;
 15    13'b0000101000z10 : data_tmp1 = 10'b0101001000;
       13'b000010100010z : data_tmp1 = 10'b0101001000;

       13'b0000101000111 : data_tmp1 = 10'b0101001001;
       13'b00001010010zz : data_tmp1 = 10'b0101001001;
 20    13'b0000101001100 : data_tmp1 = 10'b0101001001;

       13'b00001010011z1 : data_tmp1 = 10'b0101001010;
       13'b0000101001110 : data_tmp1 = 10'b0101001010;
       13'b000010101000z : data_tmp1 = 10'b0101001010;
 25
       13'b0000101010z1z : data_tmp1 = 10'b0101001011;
       13'b000010101010z : data_tmp1 = 10'b0101001011;

       13'b00001010110zz : data_tmp1 = 10'b0101001100;
 30    13'b000010101110z : data_tmp1 = 10'b0101001100;
       13'b0000101011110 : data_tmp1 = 10'b0101001100;

       13'b0000101011111 : data_tmp1 = 10'b0101001101;
       13'b00001011000zz : data_tmp1 = 10'b0101001101;
 35    13'b0000101100100 : data_tmp1 = 10'b0101001101;

       13'b00001011001z1 : data_tmp1 = 10'b0101001110;
       13'b0000101100110 : data_tmp1 = 10'b0101001110;
       13'b000010110100z : data_tmp1 = 10'b0101001110;
 40    13'b0000101101010 : data_tmp1 = 10'b0101001110;

       13'b0000101101z11 : data_tmp1 = 10'b0101001111;
       13'b000010110110z : data_tmp1 = 10'b0101001111;
       13'b0000101101110 : data_tmp1 = 10'b0101001111;
 45    13'b0000101110000 : data_tmp1 = 10'b0101001111;

       13'b00001011100z1 : data_tmp1 = 10'b0101010000;
       13'b0000101110z10 : data_tmp1 = 10'b0101010000;
       13'b000010111010z : data_tmp1 = 10'b0101010000;
 50    13'b0000101110111 : data_tmp1 = 10'b0101010000;

       13'b00001011110zz : data_tmp1 = 10'b0101010001;
       13'b000010111110z : data_tmp1 = 10'b0101010001;
       13'b0000101111110 : data_tmp1 = 10'b0101010001;
 55
       13'b0000101111111 : data_tmp1 = 10'b0101010010;
```

```
13'b000001110100z : data_tmp1 = 10'b0100110101;

13'b000001110101z : data_tmp1 = 10'b0100110110;
13'b000001110110z : data_tmp1 = 10'b0100110110;

13'b000001110111z : data_tmp1 = 10'b0100110111;
13'b000001111000z : data_tmp1 = 10'b0100110111;

13'b000001111001z : data_tmp1 = 10'b0100111000;
13'b000001111010z : data_tmp1 = 10'b0100111000;
13'b0000011110110 : data_tmp1 = 10'b0100111000;

13'b0000011110111 : data_tmp1 = 10'b0100111001;
13'b000001111100z : data_tmp1 = 10'b0100111001;
13'b0000011111010 : data_tmp1 = 10'b0100111001;

13'b0000011111011 : data_tmp1 = 10'b0100111010;
13'b000001111110z : data_tmp1 = 10'b0100111010;
13'b0000011111110 : data_tmp1 = 10'b0100111010;

13'b0000011111111 : data_tmp1 = 10'b0100111011;
13'b00001000000zz : data_tmp1 = 10'b0100111011;

13'b00001000001zz : data_tmp1 = 10'b0100111100;
13'b0000100001000 : data_tmp1 = 10'b0100111100;

13'b00001000010z1 : data_tmp1 = 10'b0100111101;
13'b0000100001010 : data_tmp1 = 10'b0100111101;
13'b0000100001100 : data_tmp1 = 10'b0100111101;

13'b000010000011z1 : data_tmp1 = 10'b0100111110;
13'b0000100001110 : data_tmp1 = 10'b0100111110;
13'b000010001000z : data_tmp1 = 10'b0100111110;

13'b000010001001z : data_tmp1 = 10'b0100111111;
13'b000010001010z : data_tmp1 = 10'b0100111111;
13'b0000100010110 : data_tmp1 = 10'b0100111111;

13'b0000100010111 : data_tmp1 = 10'b0101000000;
13'b00001000110zz : data_tmp1 = 10'b0101000000;

13'b00001000111zz : data_tmp1 = 10'b0101000001;
13'b0000100100000 : data_tmp1 = 10'b0101000001;

13'b00001001000z1 : data_tmp1 = 10'b0101000010;
13'b0000100100010 : data_tmp1 = 10'b0101000010;
13'b0000100100010z : data_tmp1 = 10'b0101000010;

13'b000010010011z : data_tmp1 = 10'b0101000011;
13'b000010010100z : data_tmp1 = 10'b0101000011;
13'b0000100101010 : data_tmp1 = 10'b0101000011;

13'b0000100101z11 : data_tmp1 = 10'b0101000100;
13'b000010010110z : data_tmp1 = 10'b0101000100;
13'b0000100101110 : data_tmp1 = 10'b0101000100;
13'b0000100110000 : data_tmp1 = 10'b0101000100;
```

```
        13'b0000001010100z : data_tmp1 = 10'b0100100011;
        13'b00000010101010 : data_tmp1 = 10'b0100100011;

        13'b00000010101011 : data_tmp1 = 10'b0100100100;
5       13'b000001010110z : data_tmp1 = 10'b0100100100;

        13'b0000001010111z : data_tmp1 = 10'b0100100101;
        13'b00000010110000 : data_tmp1 = 10'b0100100101;

10      13'b00000101100z1 : data_tmp1 = 10'b0100100110;
        13'b0000010110010 : data_tmp1 = 10'b0100100110;

        13'b000001011010z : data_tmp1 = 10'b0100100111;
        13'b0000010110110 : data_tmp1 = 10'b0100100111;
15
        13'b00000010110111 : data_tmp1 = 10'b0100101000;
        13'b000001011100z : data_tmp1 = 10'b0100101000;

        13'b000001011101z : data_tmp1 = 10'b0100101001;
20      13'b000001011110z : data_tmp1 = 10'b0100101001;

        13'b000001011111z : data_tmp1 = 10'b0100101010;
        13'b0000011000000 : data_tmp1 = 10'b0100101010;

25      13'b00000110000z1 : data_tmp1 = 10'b0100101011;
        13'b0000011000010 : data_tmp1 = 10'b0100101011;

        13'b000000110001zz : data_tmp1 = 10'b0100101100;

30      13'b000001100100z : data_tmp1 = 10'b0100101101;
        13'b0000011001010 : data_tmp1 = 10'b0100101101;

        13'b0000011001011 : data_tmp1 = 10'b0100101110;
        13'b000001100110z : data_tmp1 = 10'b0100101110;
35      13'b0000011001110 : data_tmp1 = 10'b0100101110;

        13'b0000011001111 : data_tmp1 = 10'b0100101111;
        13'b000001101000z : data_tmp1 = 10'b0100101111;
        13'b0000011010010 : data_tmp1 = 10'b0100101111;
40
        13'b0000011010011 : data_tmp1 = 10'b0100110000;
        13'b000001101010z : data_tmp1 = 10'b0100110000;

        13'b000001101011z : data_tmp1 = 10'b0100110001;
45      13'b000001101100z : data_tmp1 = 10'b0100110001;

        13'b000001101101z : data_tmp1 = 10'b0100110010;
        13'b000001101110z : data_tmp1 = 10'b0100110010;

50      13'b000001101111z : data_tmp1 = 10'b0100110011;
        13'b000001110000z : data_tmp1 = 10'b0100110011;

        13'b000001110001z : data_tmp1 = 10'b0100110100;
        13'b000001110010z : data_tmp1 = 10'b0100110100;
55
        13'b000001110011z : data_tmp1 = 10'b0100110101;
```

```
     13'b000000111010z : data_tmp1 = 10'b0100001110;

     13'b000000111011z : data_tmp1 = 10'b0100001111;

 5   13'b000000111100z : data_tmp1 = 10'b0100010000;

     13'b000000111101z : data_tmp1 = 10'b0100010001;

     13'b000000111110z : data_tmp1 = 10'b0100010010;
10   13'b0000001111110 : data_tmp1 = 10'b0100010010;

     13'b0000001111111 : data_tmp1 = 10'b0100010011;
     13'b0000010000000 : data_tmp1 = 10'b0100010011;

15   13'b0000010000001 : data_tmp1 = 10'b0100010100;
     13'b0000010000010 : data_tmp1 = 10'b0100010100;

     13'b0000010000011 : data_tmp1 = 10'b0100010101;
     13'b0000010000100 : data_tmp1 = 10'b0100010101;
20
     13'b00000100001z1 : data_tmp1 = 10'b0100010110;
     13'b0000010000110 : data_tmp1 = 10'b0100010110;

     13'b000001000100z : data_tmp1 = 10'b0100010111;
25
     13'b000001000101z : data_tmp1 = 10'b0100011000;
     13'b0000010001100 : data_tmp1 = 10'b0100011000;

     13'b0000010001101 : data_tmp1 = 10'b0100011001;
30   13'b0000010001110 : data_tmp1 = 10'b0100011001;

     13'b0000010001111 : data_tmp1 = 10'b0100011010;
     13'b000001001000z : data_tmp1 = 10'b0100011010;

35   13'b000001001001z : data_tmp1 = 10'b0100011011;

     13'b000001001010z : data_tmp1 = 10'b0100011100;
     13'b0000010010110 : data_tmp1 = 10'b0100011100;

40   13'b0000010010111 : data_tmp1 = 10'b0100011101;
     13'b000001001100z : data_tmp1 = 10'b0100011101;

     13'b000001001101z : data_tmp1 = 10'b0100011110;

45   13'b000001001110z : data_tmp1 = 10'b0100011111;
     13'b0000010011110 : data_tmp1 = 10'b0100011111;

     13'b0000010011111 : data_tmp1 = 10'b0100100000;
     13'b000001010000z : data_tmp1 = 10'b0100100000;
50
     13'b000001010001z : data_tmp1 = 10'b0100100001;
     13'b0000010100100 : data_tmp1 = 10'b0100100001;

     13'b00000101001z1 : data_tmp1 = 10'b0100100010;
55   13'b0000010100110 : data_tmp1 = 10'b0100100010;
```

```
        13'b0000000110 :  210 : data_tmp1 = 10'b0011110100;

        13'b0000001001011 : data_tmp1 = 10'b0011110101;

5       13'b000000100110z : data_tmp1 = 10'b0011110110;

        13'b0000001001110 : data_tmp1 = 10'b0011110111;

        13'b0000001001111 : data_tmp1 = 10'b0011111000;
10
        13'b000000101000z : data_tmp1 = 10'b0011111001;

        13'b0000001010010 : data_tmp1 = 10'b0011111010;

15      13'b0000001010011 : data_tmp1 = 10'b0011111011;
        13'b0000001010100 : data_tmp1 = 10'b0011111011;

        13'b0000001010101 : data_tmp1 = 10'b0011111100;

20      13'b000000101011z : data_tmp1 = 10'b0011111101;

        13'b0000001011000 : data_tmp1 = 10'b0011111110;

        13'b0000001011001 : data_tmp1 = 10'b0011111111;
25      13'b0000001011010 : data_tmp1 = 10'b0011111111;

        13'b0000001011011 : data_tmp1 = 10'b0100000000;

        13'b000000101110z : data_tmp1 = 10'b0100000001;
30
        13'b000000101111z : data_tmp1 = 10'b0100000010;

        13'b0000001100000 : data_tmp1 = 10'b0100000011;

35      13'b0000001100001 : data_tmp1 = 10'b0100000100;
        13'b0000001100010 : data_tmp1 = 10'b0100000100;

        13'b0000001100011 : data_tmp1 = 10'b0100000101;
        13'b0000001100100 : data_tmp1 = 10'b0100000101;
40
        13'b0000001100101 : data_tmp1 = 10'b0100000110;
        13'b0000001100110 : data_tmp1 = 10'b0100000110;

        13'b0000001100111 : data_tmp1 = 10'b0100000111;
45
        13'b000000110100z : data_tmp1 = 10'b0100001000;

        13'b000000110101z : data_tmp1 = 10'b0100001001;

50      13'b000000110110z : data_tmp1 = 10'b0100001010;

        13'b000000110111z : data_tmp1 = 10'b0100001011;

        13'b000000111000z : data_tmp1 = 10'b0100001100;
55
        13'b000000111001z : data_tmp1 = 10'b0100001101;
```

13'b0000000101100 : data_tmp1 = 10'b0011010111;

13'b0000000101101 : data_tmp1 = 10'b0011011000;

5   13'b0000000101110 : data_tmp1 = 10'b0011011001;

13'b0000000101111 : data_tmp1 = 10'b0011011010;

13'b0000000110000 : data_tmp1 = 10'b0011011100;

10   13'b0000000110001 : data_tmp1 = 10'b0011011101;

13'b0000000110010 : data_tmp1 = 10'b0011011110;

15   13'b0000000110011 : data_tmp1 = 10'b0011011111;

13'b0000000110100 : data_tmp1 = 10'b0011100000;

13'b0000000110101 : data_tmp1 = 10'b0011100001;

20   13'b0000000110110 : data_tmp1 = 10'b0011100010;

13'b0000000110111 : data_tmp1 = 10'b0011100011;

25   13'b0000000111000 : data_tmp1 = 10'b0011100100;

13'b0000000111001 : data_tmp1 = 10'b0011100101;

13'b0000000111010 : data_tmp1 = 10'b0011100110;

30   13'b0000000111011 : data_tmp1 = 10'b0011100111;

13'b0000000111100 : data_tmp1 = 10'b0011101000;

35   13'b0000000111101 : data_tmp1 = 10'b0011101001;

13'b0000000111110 : data_tmp1 = 10'b0011101010;

13'b0000000111111 : data_tmp1 = 10'b0011101011;

40   13'b0000001000000 : data_tmp1 = 10'b0011101100;

13'b0000001000001 : data_tmp1 = 10'b0011101101;

45   13'b0000001000010 : data_tmp1 = 10'b0011101110;

13'b0000001000011 : data_tmp1 = 10'b0011101111;
13'b0000001000100 : data_tmp1 = 10'b0011101111;

50   13'b0000001000101 : data_tmp1 = 10'b0011110000;

13'b0000001000110 : data_tmp1 = 10'b0011110001;

13'b0000001000111 : data_tmp1 = 10'b0011110010;

55   13'b000000100100z : data_tmp1 = 10'b0011110011;

```
      13'b0000000010000 : data_tmp1 = 10'b0010011101;

      13'b0000000010001 : data_tmp1 = 10'b0010100001;

  5   13'b0000000010010 : data_tmp1 = 10'b0010100100;

      13'b0000000010011 : data_tmp1 = 10'b0010100111;

      13'b0000000010100 : data_tmp1 = 10'b0010101010;
 10
      13'b0000000010101 : data_tmp1 = 10'b0010101101;

      13'b0000000010110 : data_tmp1 = 10'b0010101111;

 15   13'b0000000010111 : data_tmp1 = 10'b0010110010;

      13'b0000000011000 : data_tmp1 = 10'b0010110100;

      13'b0000000011001 : data_tmp1 = 10'b0010110111;
 20
      13'b0000000011010 : data_tmp1 = 10'b0010111001;

      13'b0000000011011 : data_tmp1 = 10'b0010111011;

 25   13'b0000000011100 : data_tmp1 = 10'b0010111101;

      13'b0000000011101 : data_tmp1 = 10'b0010111111;

      13'b0000000011110 : data_tmp1 = 10'b0011000001;
 30
      13'b0000000011111 : data_tmp1 = 10'b0011000011;

      13'b0000000100000 : data_tmp1 = 10'b0011000101;

 35   13'b0000000100001 : data_tmp1 = 10'b0011000110;

      13'b0000000100010 : data_tmp1 = 10'b0011001000;

      13'b0000000100011 : data_tmp1 = 10'b0011001010;
 40
      13'b0000000100100 : data_tmp1 = 10'b0011001011;

      13'b0000000100101 : data_tmp1 = 10'b0011001101;

 45   13'b0000000100110 : data_tmp1 = 10'b0011001110;

      13'b0000000100111 : data_tmp1 = 10'b0011010000;

      13'b0000000101000 : data_tmp1 = 10'b0011010001;
 50
      13'b0000000101001 : data_tmp1 = 10'b0011010011;

      13'b0000000101010 : data_tmp1 = 10'b0011010100;

 55   13'b0000000101011 : data_tmp1 = 10'b0011010101;
```

113

variance of the data and hence the F_ratio.

Notes :

`***********************************************************************/`

`timescale 1ns / 100ps`

```verilog
module fft_window_lu (clk, enable_3, in_address, out_data);

parameter        r_wordlength = 10;   // Data wordlength.
parameter        lu_AddressSize = 13;   // Address bus size.

input        clk,
             enable_3;
input [lu_AddressSize-1:0] in_address;

output [r_wordlength-1:0] out_data;

reg  [r_wordlength-1:0] data_tmp1,
             data_tmp2;

always @(in_address)
casez (in_address)
13'b0000000000000 : data_tmp1 = 10'b1000000000;

13'b0000000000001 : data_tmp1 = 10'b0000000000;

13'b0000000000010 : data_tmp1 = 10'b0000100111;

13'b0000000000011 : data_tmp1 = 10'b0000111110;

13'b0000000000100 : data_tmp1 = 10'b0001001110;

13'b0000000000101 : data_tmp1 = 10'b0001011011;

13'b0000000000110 : data_tmp1 = 10'b0001100110;

13'b0000000000111 : data_tmp1 = 10'b0001101110;

13'b0000000001000 : data_tmp1 = 10'b0001110110;

13'b0000000001001 : data_tmp1 = 10'b0001111101;

13'b0000000001010 : data_tmp1 = 10'b0010000011;

13'b0000000001011 : data_tmp1 = 10'b0010001000;

13'b0000000001100 : data_tmp1 = 10'b0010001101;

13'b0000000001101 : data_tmp1 = 10'b0010010001;

13'b0000000001110 : data_tmp1 = 10'b0010010110;

13'b0000000001111 : data_tmp1 = 10'b0010011010;
```

```
      ovf_10 || ovf_11 || ovf_12 || ovf_13 || ovf_14 ||
      ovf_15;

   // 2k/8k Overflow flag configuration.
5  always @(in_2k8k or ovf_16 or ovf_17 or ovf_18 or ovf_2k)
    if (in_2k8k)
     ovf_tmp1 = ovf_2k || ovf_16 || ovf_17 || ovf_18;
    else
     ovf_tmp1 = ovf_2k;
10
   always @(posedge clk)           // Register overflow
    if (enable_3 && fft_cycle_complete)      // flag to change when
     ovf_tmp2 <= ovf_tmp1;          // I/Q samples are valid
                  // from FFT processor.
15 assign out_ovf = ovf_tmp2;

   `ifdef OVERFLOW_DEBUG
   // Debug code to display overflow output of a particular instance.
   // Concurrently monitor overflow flag and halt on overflow.
20 always @(out_ovf) // ovf_x wires are all registered at lower level.
    if (out_ovf)
     begin
      $display ("Overflow has occurred, type . to continue.");
      $display ("Overflow flag, out_ovf = ",out_ovf);
25    if (ovf_18) $display ("Overflow on port ovf_18");
      if (ovf_17) $display ("Overflow on port ovf_17");
      if (ovf_16) $display ("Overflow on port ovf_16");
      if (ovf_15) $display ("Overflow on port ovf_15");
      if (ovf_14) $display ("Overflow on port ovf_14");
30    if (ovf_13) $display ("Overflow on port ovf_13");
      if (ovf_12) $display ("Overflow on port ovf_12");
      if (ovf_11) $display ("Overflow on port ovf_11");
      if (ovf_10) $display ("Overflow on port ovf_10");
      if (ovf_9) $display ("Overflow on port ovf_9");
35    if (ovf_8) $display ("Overflow on port ovf_8");
      if (ovf_7) $display ("Overflow on port ovf_7");
      if (ovf_6) $display ("Overflow on port ovf_6");
      if (ovf_5) $display ("Overflow on port ovf_5");
      if (ovf_4) $display ("Overflow on port ovf_4");
40    if (ovf_3) $display ("Overflow on port ovf_3");
      if (ovf_2) $display ("Overflow on port ovf_2");
      if (ovf_1) $display ("Overflow on port ovf_1");
      if (ovf_0) $display ("Overflow on port ovf_0");
      $stop;
45   end
   `endif
   endmodule
```

Listing 13

```
50 // SccsId: %W% %G%
   /******************************************************************
```

Copyright (c) 1997 Pioneer Digital Design Centre Limited

Author   : Dawood Alam.

55

Description: Verilog code for the window lookup table, used to determine the

```verilog
      t <= t + 1'b1;

      assign ram_address = t;

5     assign ram_enable = enable_3 || enable_2;     // ram enable signal.
      // ------------------------------------------------------------
      //        valid_out status flag generation.
      // ------------------------------------------------------------

10    always @(posedge clk)
       if (!nrst)
        fft_cycle_complete <= 1'b0; // Detect end of 1st fft cycle i.e. 2K or 8K.
        else if ((~in_2k8k && &address[10:0]) || (in_2k8k && &address[12:0]))
        fft_cycle_complete <= 1'b1;
15     else
        fft_cycle_complete <= fft_cycle_complete;

      always @(posedge clk)       // Account for pipeline and I/O registers.
       if (!nrst)
20      pipeline_count <= 4'b0;   // Stop at pipeline_depth - 1.
        else if (enable_3 && fft_cycle_complete & pipeline_count < 8)//pipe depth=8
         pipeline_count <= pipeline_count + 1'b1;

      always @(posedge clk)       // Test if the pipeline is full and the input
25     if (!nrst)        // is valid before asserting valid_out.
        output_valid <= 1'b0;
        else if (enable_2 && pipeline_count[3])
         output_valid <= 1'b1;
        else
30      output_valid <= 1'b0;

      assign valid_out = output_valid;

      // ------------------------------------------------------------
35    //     Fast 40 MHz clock decoder and valid_in control.
      // ------------------------------------------------------------

      always @(posedge clk)
       if (!nrst)            // Synchronous power-up reset.
40      r <= 0;
        else if (valid_in)         // Count if input data valid.
         r <= r + 1'b1;

      assign control = {valid_in & r[1],valid_in & r[0]};
45
      assign enable_0 = valid_in & (~r[1] & ~r[0]);   // Gate valid_in with
      assign enable_1 = valid_in & (~r[1] & r[0]);   // decoded enable signals
      assign enable_2 = valid_in & ( r[1] & ~r[0]);   // to control all reg's.
      assign enable_3 = valid_in & ( r[1] & r[0]);
50
      // ------------------------------------------------------------
      // Overflow detection, OR overflows from each stage to give overflow flag.
      // ------------------------------------------------------------

55    assign ovf_2k = ovf_0 || ovf_1 || ovf_2 || ovf_3 || ovf_4 ||
            ovf_5 || ovf_6 || ovf_7 || ovf_8 || ovf_9 ||
```

```
assign out_xr = xr_tmp2;
assign out_xi = xi_tmp2;

always @(posedge clk)        // RAMs are latched on outputs so no
begin              // need to enable.
  z2r_4 <= z2r_4_tmp;        // Register FFT outputs to RAM.
  z2i_4 <= z2i_4_tmp;
  z2r_5 <= z2r_5_tmp;
  z2i_5 <= z2i_5_tmp;
  z2r_6 <= z2r_6_tmp;
  z2i_6 <= z2i_6_tmp;
  z2r_7 <= z2r_7_tmp;
  z2i_7 <= z2i_7_tmp;
  z2r_8 <= z2r_8_tmp;
  z2i_8 <= z2i_8_tmp;
  z2r_9 <= z2r_9_tmp;
  z2i_9 <= z2i_9_tmp;
//  z2r_10 <= z2r_10_tmp;
//  z2i_10 <= z2i_10_tmp;
  x1r_4_tmp <= x1r_4;      // Register FFT inputs from RAM.
  x1i_4_tmp <= x1i_4;
  x1r_5_tmp <= x1r_5;
  x1i_5_tmp <= x1i_5;
  x1r_6_tmp <= x1r_6;
  x1i_6_tmp <= x1i_6;
  x1r_7_tmp <= x1r_7;
  x1i_7_tmp <= x1i_7;
  x1r_8_tmp <= x1r_8;
  x1i_8_tmp <= x1i_8;
  x1r_9_tmp <= x1r_9;
  x1i_9_tmp <= x1i_9;
//  x1r_10_tmp <= x1r_10;
//  x1i_10_tmp <= x1i_10;
end


// ----------------------------------------------------------
//        Synchronous butterfly controller.
// ----------------------------------------------------------

always @(posedge clk)
  if (!nrst)            // Synchronous power-up reset.
    q <= 0;
  else if (enable_3)
    q <= q + 1'b1;

assign address = q;


// ----------------------------------------------------------
//        Synchronous RAM address generator.
// ----------------------------------------------------------

always @(posedge clk)
  if (!nrst)            // Synchronous power-up reset.
    t <= 0;
  else if (enable_2)
```

```
            ovf_1);

      fft_bf2l #(wordlength) bf2l_0 (clk, enable_1,
                  x1r_0, x1i_0,        // Inputs.
5                   x2r_0, x2i_0,
                  s[0],
                  xr_tmp1, xi_tmp1,       // Outputs.
                    z2r_0, z2i_0,
                  ovf_0);
10
      assign s[0] = ~address[0]; // Invert s[0] (see count sequence), SR0 not req..
      //fft_sr_1bit #(7) sr_1bit_0 (clk, enable_3, address[0], s[0]); // SR 0.

      // Last stage should be just a single register as only 1 location needed.
15    always @(posedge clk) // No reset required as data clocked through registers.
       if (enable_3)
         begin
           x1r_0_reg <= z2r_0;
           x1i_0_reg <= z2i_0;
20       end

      assign x1r_0 = x1r_0_reg;
      assign x1i_0 = x1i_0_reg;

25    // -----------------------------------------------------------
      //          Register Inputs/Outputs.
      // -----------------------------------------------------------

      `ifdef BIN_SHIFT
30    always @(posedge clk)      // Registered inputs.
       if (enable_3 && !address[0])   // == freq bin shift by pi.
         begin
           xr_reg <= in_xr;
           xi_reg <= in_xi;
35       end
        else if (enable_3 && address[0]) // == freq bin shift by pi.
         begin
           xr_reg <= ~in_xr + 1'b1;   // This is equivalent to multiplying by
           xi_reg <= ~in_xi + 1'b1;   // exp(-j * pi * n) == (-1)^n.
40       end
      `else
      always @(posedge clk)       // Registered inputs.
       if (enable_3)
         begin
45         xr_reg <= in_xr;
           xi_reg <= in_xi;
         end
      `endif

50    always @(posedge clk)       // Registered outputs.
       if (enable_3)
         begin
           xr_tmp2 <= xr_tmp1;
           xi_tmp2 <= xi_tmp1;
55       end
```

```
                z2r_3, z2i_3,
                ovf_5);

        fft_sr_1bit #(5) sr_1bit_3 (clk, enable_3, address[3], s[3]); // SR 3.
5       fft_sr_1bit #(5) sr_1bit_4 (clk, enable_3, address[4], s[4]); // SR 4.

        fft_sr_iq #(wordlength, 8) sr_iq_3 (clk, enable_3,    // Length = 8.
                z2r_3, z2i_3,     // Inputs.
                x1r_3, x1i_3);    // Outputs.
10
        // ------------------------------------------------------------
        //              Section 2 and 1.
        // ------------------------------------------------------------

15      fft_complex_mult_mux #(wordlength, c_wordlength, mult_scale) m1
                (clk, control,
                    ar_1, ai_1, br_1, bi_1,    // Inputs.
                x2r_2, x2i_2,             // Outputs.
                ovf_4);
20
        fft_bf2I #(wordlength) bf2I_1 (clk, enable_1,
                x1r_2, x1i_2,      // Inputs.
                x2r_2, x2i_2,
                s[2],
25              x2r_1, x2i_1,      // Outputs.
                z2r_2, z2i_2,
                ovf_3);

        fft_sr_iq #(wordlength, 4) sr_iq_2 (clk, enable_3,    // Length = 4.
30              z2r_2, z2i_2,     // Inputs.
                x1r_2, x1i_2);    // Outputs.

        fft_bf2II #(wordlength) bf2II_1 (clk, enable_1,
                x1r_1, x1i_1,      // Inputs.
35              x2r_1, x2i_1,
                s[1], s[2],
                ar_0, ai_0,        // Outputs.
                z2r_1, z2i_1,
                ovf_2);
40
        assign s[1] = ~address[1]; // Invert s[1] (see count sequence), SR1 not req.
        //fft_sr_1bit #(6) sr_1bit_1 (clk, enable_3, address[1], s[1]); // SR 1.
        fft_sr_1bit #(6) sr_1bit_2 (clk, enable_3, address[2], s[2]); // SR 2.

45      fft_sr_iq #(wordlength, 2) sr_iq_1 (clk, enable_3,    // Length = 2.
                z2r_1, z2i_1,     // Inputs.
                x1r_1, x1i_1);    // Outputs.


        // ------------------------------------------------------------
50      //  Section 0, front end of FFT pipeline (output stage), mult_scale=4.
        // ------------------------------------------------------------

        fft_complex_mult_mux #(wordlength, c_wordlength, 4) m0
                (clk, control,
55                  ar_0, ai_0, br_0, bi_0,    // Inputs.
                x2r_0, x2i_0,             // Outputs.
```

```
fft_sr_1bit #(3) sr_1bit_7 (clk, enable_3, address[7], s[7]); // SR 7.
fft_sr_1bit #(3) sr_1bit_8 (clk, enable_3, address[8], s[8]); // SR 8.

// --------------------------------------------------------
//              Section 6 and 5.
// --------------------------------------------------------

fft_complex_mult_mux #(wordlength, c_wordlength, mult_scale) m3
        (clk, control,
            ar_3, ai_3, br_3, bi_3,     // Inputs.
        x2r_6, x2i_6,          // Outputs.
        ovf_10);

fft_bf2I #(wordlength) bf2I_3 (clk, enable_1,
            x1r_6_tmp, x1i_6_tmp,    // inputs.
            x2r_6, x2i_6,
            s[6],
            x2r_5, x2i_5,        // Outputs.
            z2r_6_tmp, z2i_6_tmp,
            ovf_9);

fft_bf2II #(wordlength) bf2II_3 (clk, enable_1,
            x1r_5_tmp, x1i_5_tmp,    // Inputs.
            x2r_5, x2i_5,
            s[5], s[6],
            ar_2, ai_2,         // Outputs.
            z2r_5_tmp, z2i_5_tmp,
            ovf_8);

fft_sr_1bit #(4) sr_1bit_5 (clk, enable_3, address[5], s[5]); // SR 5.
fft_sr_1bit #(4) sr_1bit_6 (clk, enable_3, address[6], s[6]); // SR 6.

// --------------------------------------------------------
//              Section 4 and 3.
// --------------------------------------------------------

fft_complex_mult_mux #(wordlength, c_wordlength, mult_scale) m2
        (clk, control,
            ar_2, ai_2, br_2, bi_2,     // Inputs.
        x2r_4, x2i_4,          // Outputs.
        ovf_7);

fft_bf2I #(wordlength) bf2I_2 (clk, enable_1,
            x1r_4_tmp, x1i_4_tmp,    // Inputs.
            x2r_4, x2i_4,
            s[4],
            x2r_3, x2i_3,        // Outputs.
            z2r_4_tmp, z2i_4_tmp,
            ovf_6);

fft_bf2II #(wordlength) bf2II_2 (clk, enable_1,
            x1r_3, x1i_3,        // Inputs.
            x2r_3, x2i_3,
            s[3], s[4],
            ar_1, ai_1,         // Outputs.
```

```
// ----------------------------------------------------------------
//                Section 10 and 9.
// ----------------------------------------------------------------

fft_complex_mult_mux #(wordlength, c_wordlength, mult_scale) m5
        (clk, control,
            ar_5, ai_5, br_5, bi_5,      // Inputs.
            x2r_10_tmp1, x2i_10_tmp1,       // Outputs.
            ovf_16);

fft_bf2I #(wordlength) bf2I_5 (clk, enable_1,
            x1r_10, x1i_10      // Inputs.
            x2r_10, x2i_10,
            s[10],
            x2r_9, x2i_9,          // Outputs.
            z2r_10, z2i_10,
            ovf_15);

fft_bf2II #(wordlength) bf2II_5 (clk, enable_1,
            x1r_9_tmp, x1i_9_tmp,     // Inputs.
            x2r_9, x2i_9,
            s[9], s[10],
            ar_4, ai_4,          // Outputs.
            z2r_9_tmp, z2i_9_tmp,
            ovf_14);

fft_sr_1bit #(2) sr_1bit_9 (clk, enable_3, address[9], s[9]); // SR 9.
fft_sr_1bit #(2) sr_1bit_10 (clk, enable_3, address[10], s[10]); // SR 10.


// ----------------------------------------------------------------
//                Section 8 and 7.
// ----------------------------------------------------------------

fft_complex_mult_mux #(wordlength, c_wordlength, mult_scale) m4
        (clk, control,
            ar_4, ai_4, br_4, bi_4,      // Inputs.
            x2r_8, x2i_8,           // Outputs.
            ovf_13);

fft_bf2I #(wordlength) bf2I_4 (clk, enable_1,
            x1r_8_tmp, x1i_8_tmp,     // Inputs.
            x2r_8, x2i_8,
            s[8],
            x2r_7, x2i_7,          // Outputs.
            z2r_8_tmp, z2i_8_tmp,
            ovf_12);

fft_bf2II #(wordlength) bf2II_4 (clk, enable_1,
            x1r_7_tmp, x1i_7_tmp,     // Inputs.
            x2r_7, x2i_7,
            s[7], s[8],
            ar_3, ai_3,          // Outputs.
            z2r_7_tmp, z2i_7_tmp,
            ovf_11);
```

```
begin
    x2r_10_tmp2 = x2r_10_tmp1;
    x2i_10_tmp2 = x2i_10_tmp1;
    // Sign extend from 10 bits, as section 12 is s12_wdlength bits.
    in_xr_tmp = {{(s12_wdlength-9){xr_reg[9]}},xr_reg[8:0]};
    in_xi_tmp = {{(s12_wdlength-9){xi_reg[9]}},xi_reg[8:0]};
end


always @(posedge clk) // Pipeline register to enable correct operation in
    if (enable_3)    // 2K mode without retiming the entire pipeline since
    begin       // 8K mode introduces 1 additional pipeline register.
        // Sign extend 10 bit inputs to wordlength bit inputs.
        // for bypass lines into stage 5.
        x2r_10_tmp3 <= {{(wordlength-9){xr_reg[9]}},xr_reg[8:0]};
        x2i_10_tmp3 <= {{(wordlength-9){xi_reg[9]}},xi_reg[8:0]};
    end


assign x2r_10 = x2r_10_tmp2;
assign x2i_10 = x2i_10_tmp2;


// Sign extend from s12_wdlength bits to s11_wdlength bits between
// sections 12 and 11. Uncomment below if s_11 < > s_12.
assign x2r_11 = {{(s11_wdlength-s12_wdlength+1)
        {x2r_11_tmp[s12_wdlength-1]}},x2r_11_tmp[s12_wdlength-2:0]};
assign x2i_11 = {{(s11_wdlength-s12_wdlength+1)
        {x2i_11_tmp[s12_wdlength-1]}},x2i_11_tmp[s12_wdlength-2:0]};
// Uncomment below if s_11 = s_12.
/* assign x2r_11 = x2r_11_tmp;
   assign x2i_11 = x2i_11_tmp; */


fft_bf2I #(s12_wdlength) bf2I_6
        (clk, enable_1,
            x1r_12, x1i_12, in_xr_tmp, in_xi_tmp,  // Ext In.
        s[12],
        x2r_11_tmp, x2i_11_tmp, z2r_12, z2i_12, // Outputs.
        ovf_18);


/* fft_ram #(s12_wdlength, 12) ram_12 (clk, enable_1, enable_3,
            ram_address[11:0],     // 4096 addrs.
            z2r_12, z2i_12,     // Inputs.
            x1r_12, x1i_12);       // Outputs. */


fft_bf2II #(s11_wdlength) bf2II_6
        (clk, enable_1,
            x1r_11, x1i_11, x2r_11, x2i_11,     // Inputs.
        s[11], s[12],
        ar_5, ai_5, z2r_11, z2i_11,       // Outputs.
        ovf_17);


fft_sr_1bit #(1) sr_1bit_11 (clk, enable_3, address[11], s[11]); // SR 11.
fft_sr_1bit #(1) sr_1bit_12 (clk, enable_3, address[12], s[12]); // SR 12.


/* fft_ram #(s11_wdlength, 11) ram_11 (clk, enable_1, enable_3,
            ram_address[10:0],     // 2048 addrs.
            z2r_11, z2i_11,     // Inputs.
            x1r_11, x1i_11);       // Outputs. */
```

```
       fft_hardwired_lu2 #(c_wordlength, rom_AddressSize-6) // Case table instance
       rom2 (clk, enable_3, address[6:0], br_2, bi_2);   // for a hardwired ROM.

       /*fft_hardwired_lu3 #(c_wordlength, rom_AddressSize-4) // Case table instance
  5    rom3 (clk, enable_3, address[8:0], br_3, bi_3);   // for a hardwired ROM.*/

       /*fft_hardwired_lu3 #(c_wordlength, rom_AddressSize-5)// Case table instance
       rom3 (clk, enable_3, address_rom3, br_3, bi_3);   // for a hardwired ROM.*/

 10    /*fft_rom #(c_wordlength, rom_AddressSize-6,
          "../../../fft/src/lookup_tables/lu_10bit_128pt_scale1")
       rom2 (address[6:0], br_2, bi_2); // 128 addresses x 20 bits, no decode. */

       /*fft_rom #(c_wordlength, rom_AddressSize-7,
 15       "../../../fft/src/lookup_tables/lu_10bit_128pt_scale1")
       rom2 (address_rom2, br_2, bi_2); // 64 addresses x 20 bits, coeff decode. */

       /*fft_rom #(c_wordlength, rom_AddressSize-4,
          "../../../fft/src/lookup_tables/lu_10bit_512pt_scale1")
 20    rom3 (address[8:0], br_3, bi_3); // 512 addresses x 20 bits, no decode. */

       /* fft_rom #(c_wordlength, rom_AddressSize-5,
          "../../../fft/src/lookup_tables/lu_10bit_512pt_scale1")
       rom3 (clk, enable_3, address_rom3, br_3, bi_3); // 256 addresses x 20 bits.*/
 25
       /*fft_rom #(c_wordlength, rom_AddressSize-2,
          "../../../fft/src/lookup_tables/lu_10bit_2048pt_scale1")
       rom4 (address[10:0], br_4, bi_4); // 2048 addresses x 20 bits, no decode. */

 30    /* fft_rom #(c_wordlength, rom_AddressSize-3,
          "../../../fft/src/lookup_tables/lu_10bit_2048pt_scale1")
       rom4 (clk, enable_3, address_rom4, br_4, bi_4); // 1024 addresses x 20 bits.*/

       /*fft_rom #(c_wordlength, rom_AddressSize,
 35       "../../../fft/src/lookup_tables/lu_10bit_8192pt_scale1")
       rom5 (address, br_5, bi_5); // 8192 addresses x 20 bits, no decode. */

       /* fft_rom #(c_wordlength, rom_AddressSize-1,
          "../../../fft/src/lookup_tables/lu_10bit_8192pt_scale1")
 40    rom5 (clk, enable_3, address_rom5, br_5, bi_5); // 4096 addresses x 20 bits.*/

       // ------------------------------------------------------------
       //    Section 12 and 11, tail end of FFT pipeline (input stage).
       // Section 12 is 11 bits wide and incorporates the 2K/8K control logic.
 45    // ------------------------------------------------------------

       always @(xr_reg or xi_reg or in_2k8k or x2r_10_tmp1 or x2i_10_tmp1
                 or x2r_10_tmp3 or x2i_10_tmp3)
          if (!in_2k8k) // Configuring for 2K mode.
 50       begin
             x2r_10_tmp2 = x2r_10_tmp3;
             x2i_10_tmp2 = x2i_10_tmp3;
             in_xr_tmp = 0;
             in_xi_tmp = 0;
 55       end
          else    // Configuring for 8K mode.
```

```
reg  [9:0]    xr_reg,        // Input data reg, I.
         xi_reg;       // Input data reg, Q.
reg [wordlength-1:0] x2r_10_tmp2, x2i_10_tmp2,
         x2r_10_tmp3, x2i_10_tmp3;

wire [wordlength-1:0] xr_tmp1,       // Final BF2I(0) out, I.
         xi_tmp1;      // Final BF2I(0) out, Q.
wire [wordlength-1:0] x2r_10_tmp1, x2i_10_tmp1;
wire [s12_wdlength-1:0] x2r_11_tmp, x2i_11_tmp;

// -------------------------------------------------------------
//  Address decoders/Quadrant mappers + pipeline shift registers.
// -------------------------------------------------------------

/* fft_sr_addr #(rom_AddressSize-6, 3) sr_addr_2
         (clk, enable_3,
          address[6:0],  // Input.
          dcd_address2);  // Output.

fft_coeff_dcd #(rom_AddressSize-6, 11, 21)
coeff_dcd_2 (clk, enable_3, dcd_address2, address_rom2, nrst); */

// -------------------------------------------------------------

fft_sr_addr #(rom_AddressSize-4, 2) sr_addr_3
         (clk, enable_3,
          address[8:0],  // Input.
          dcd_address3);  // Output.

fft_coeff_dcd #(rom_AddressSize-4, 43, 85)
coeff_dcd_3 (clk, enable_3, dcd_address3, address_rom3, nrst);

// -------------------------------------------------------------

fft_sr_addr #(rom_AddressSize-2, 1) sr_addr_4
         (clk, enable_3,
          address[10:0],  // Input.
          dcd_address4);  // Output.

fft_coeff_dcd #(rom_AddressSize-2, 171, 341)
coeff_dcd_4 (clk, enable_3, dcd_address4, address_rom4, nrst);

// -------------------------------------------------------------

/* fft_coeff_dcd #(rom_AddressSize, 683, 1365)
   coeff_dcd_5 (clk, enable_3, address, address_rom5, nrst); */

// -------------------------------------------------------------
//          ROM lookup tables.
// -------------------------------------------------------------

fft_hardwired_lu0 #(c_wordlength, rom_AddressSize-10)// Case table instance
rom0 (clk, enable_3, address[2:0], br_0, bi_0);   // for a hardwired ROM.

fft_hardwired_lu1 #(c_wordlength, rom_AddressSize-8) // Case table instance
rom1 (clk, enable_3, address[4:0], br_1, bi_1);   // for a hardwired ROM.
```

```
wire  [wordength-1:0]  z2r_10, z2i_10;  // WILL CHANGE WHEN RAM RIGHT 2 rg

wire  [wordlength-1:0]  z2r_4_tmp, z2i_4_tmp,  // Couple the I/Q data
          z2r_5_tmp, z2i_5_tmp,  // outputs of each BF
            z2r_6_tmp, z2i_6_tmp,  // processor to their
          z2r_7_tmp, z2i_7_tmp,  // respective memory
            z2r_8_tmp, z2i_8_tmp,  // inputs via an output
            z2r_9_tmp, z2i_9_tmp,  // register.
          z2r_10_tmp, z2i_10_tmp;

wire  [s11_wdlength-1:0]  z2r_11, z2i_11;  // Different bit-widths
wire  [s12_wdlength-1:0]  z2r_12, z2i_12;  // for the 1st 2 stages.

wire  [rom_AddressSize-8:0]  address_rom2;  // Couples the address
wire  [rom_AddressSize-6:0]  address_rom3;  // decoders outputs to
wire  [rom_AddressSize-4:0]  address_rom4;  // respective ROMs.
wire  [rom_AddressSize-2:0]  address_rom5;

wire  [rom_AddressSize-7:0]  dcd_address2;  // Couples part of the
wire  [rom_AddressSize-5:0]  dcd_address3;  // address bus to the
wire  [rom_AddressSize-3:0]  dcd_address4;  // coefficient decoder.
wire  [rom_AddressSize-1:0]  dcd_address5;

wire          ovf_0, ovf_1,      // Couples overflow
          ovf_2, ovf_3,        // flag outputs from
            ovf_4, ovf_5,      // each butterfly
            ovf_6, ovf_7,      // processor and complex
            ovf_8, ovf_9,      // multiplier into one
            ovf_10, ovf_11,      // overflow status flag
            ovf_12, ovf_13,      // called "out_ovf".
            ovf_14, ovf_15,
            ovf_16, ovf_17,
            ovf_18;

wire          clk,
          nrst,
            in_2k8k,
          ovf_2k,
            out_ovf,
            enable_0,
            enable_1,
            enable_2,
            enable_3,
            ram_enable;        // RAM enable signal.

reg          ovf_tmp1,
          ovf_tmp2,
          fft_cycle_complete,    // End of 1st FFT cycle.
            output_valid;        // Output valid flag.
reg  [3:0]      pipeline_count;      // Counts pipeline regs.
reg  [AddressSize-1:0] q, t;
reg  [1:0]      r;
reg  [wordlength-1:0] x1r_0_reg, x1i_0_reg,
          xr_tmp2,        // Output data reg, I.
            xi_tmp2;        // Output data reg, Q.
reg  [s12_wdlength-1:0] in_xr_tmp, in_xi_tmp;
```

```
          x1r_5, x1i_5,      // input register.
            x1r_6, x1i_6,
            x1r_7, x1i_7,
            x1r_8, x1i_8,
  5         x1r_9, x1i_9,
            x1r_10, x1i_10,


          x2r_0, x2i_0,      // Couples the I/Q data
            x2r_1, x2i_1,      // outputs from BF2I
 10         x2r_2, x2i_2,      // to the I/Q inputs of
            x2r_3, x2i_3,      // BF2II. Also connects
          x2r_4, x2i_4,     // the I/Q ouputs of the
          x2r_5, x2i_5,     // complex multiplier
          x2r_6, x2i_6,     // to the inputs of the
 15       x2r_7, x2i_7,     // next radix 2^2 stage.
            x2r_8, x2i_8,
            x2r_9, x2i_9,
            x2r_10, x2i_10;


 20   reg [wordlength-1:0] x1r_4_tmp, x1i_4_tmp, // Registered inputs
            x1r_5_tmp, x1i_5_tmp,  // from RAM.
            x1r_6_tmp, x1i_6_tmp,
            x1r_7_tmp, x1i_7_tmp,
            x1r_8_tmp, x1i_8_tmp,
 25         x1r_9_tmp, x1i_9_tmp,
            x1r_10_tmp, x1i_10_tmp;


      wire  [s11_wdlength-1:0] x1r_11, x1i_11,    // Different bit-widths
            x2r_11, x2i_11;       // for I/Q lines, but
 30   wire  [s12_wdlength-1:0] x1r_12, x1i_12;    // similar to the above.


      wire  [wordlength-1:0] ar_0, ai_0,       // Couples the I/Q data'
            ar_1, ai_1,        // outputs of the
            ar_2, ai_2,        // previous radix 2^2
 35         ar_3, ai_3,        // stage into the
          ar_4, ai_4,      // complex multiplier
            ar_5, ai_5;        // of the next stage.


      wire  [c_wordlength-1:0] br_0, bi_0,       // Couples the I/Q
 40       br_1, bi_1,       // coefficient outputs
            br_2, bi_2,        // from the ROM demapper
            br_3, bi_3,        // to the complex
          br_4, bi_4,      // multiplier.
            br_5, bi_5;
 45
      wire  [wordlength-1:0]  z2r_0, z2i_0,
            z2r_1, z2i_1,
            z2r_2, z2i_2,
          z2r_3, z2i_3;
 50
      reg  [wordlength-1:0]  z2r_4, z2i_4,       // Registered outputs
            z2r_5, z2i_5,      // to RAM.
            z2r_6, z2i_6,
            z2r_7, z2i_7,
 55         z2r_8, z2i_8,
            z2r_9, z2i_9;
```

100

```
        input       clk,          // Master clock.
                    nrst,         // Power-up reset.
                    in_2k8k,      // 2K mode active low.
                    valid_in;     // Input data valid.
  5     input [9:0]     in_xr,        // FFT input data, I.
                    in_xi;        // FFT input data, Q.

        input [wordlength-1:0] x1r_4, x1i_4,    // RAM output ports.
                    x1r_5, x1i_5,
 10             x1r_6, x1i_6,
                    x1r_7, x1i_7,
                    x1r_8, x1i_8,
                    x1r_9, x1i_9,
                    x1r_10, x1i_10;
 15
        input [c_wordlength-1:0]  br_3, bi_3,
                    br_4, bi_4;

        output      out_ovf,      // Overflow flag.
 20             enable_0,     // Enable clock 0.
                    enable_1,     // Enable clock 1.
                    enable_2,     // Enable clock 2.
                    enable_3,     // Enable clock 3.
                    valid_out,    // Output data valid.
 25             ram_enable;

        output [wordlength-1:0] out_xr,      // FFT output data, I.
                    out_xi;       // FFT output data, Q.

 30     output [wordlength-1:0] z2r_4, z2i_4,    // RAM input ports.
                    z2r_5, z2i_5,
                    z2r_6, z2i_6,
                    z2r_7, z2i_7,
                    z2r_8, z2i_8,
 35             z2r_9, z2i_9,
                    z2r_10, z2i_10;

        output [rom_AddressSize-6:0] address_rom3;
        output [rom_AddressSize-4:0] address_rom4;
 40
        output [AddressSize-1:0] ram_address;

        // ------------------------------------------------------------
        //          Wire/register declarations.
 45     // ------------------------------------------------------------

        wire [1:0]      control;      // clk decode.
        wire [AddressSize-1:0] address,      // FFT main address bus.
                    s,            // Pipeline SRs to BFs.
 50             ram_address;      // RAM address bus.

        wire [wordlength-1:0] x1r_0, x1i_0,     // Couples the I/Q data
                    x1r_1, x1i_1,     // outputs from the
                    x1r_2, x1i_2,     // memory to the
 55             x1r_3, x1i_3,     // respective butterfly
                    x1r_4, x1i_4,     // processors, via an
```

```
`timescale 1ns / 100ps

module fft_r22sdf    (in_xr,
              in_xi,
              clk,
              nrst,
                in_2k8k,
                valid_in,
              out_xr,
                out_xi,
                out_ovf,
              enable_0,
                enable_1,
                enable_2,
                enable_3,
                valid_out,
                ram_address,
                ram_enable,
                address_rom3,
                address_rom4,
              z2r_4, z2i_4,      // RAM input ports.
                z2r_5, z2i_5,      // Output data from this
                z2r_6, z2i_6,      // module.
                z2r_7, z2i_7,
                z2r_8, z2i_8,
                z2r_9, z2i_9,
                z2r_10, z2i_10,
                x1r_4, x1i_4,      // RAM output ports.
                x1r_5, x1i_5,      // Input data to this
                x1r_6, x1i_6,      // module.
                x1r_7, x1i_7,
                x1r_8, x1i_8,
                x1r_9, x1i_9,
                x1r_10, x1i_10,
                br_3, bi_3,
                br_4, bi_4);

// --------------------------------------------------------------
//          Parameter definitions.
// --------------------------------------------------------------

parameter        wordlength = 12;    // Data wordlength.
parameter        c_wordlength = 10;    // Coeff wordlength.
parameter        AddressSize = 13;    // Size of address bus.
parameter        rom_AddressSize = 13;  // ROM address bus size.
parameter        mult_scale = 3;      // Multiplier scaling:
                 // 1 = /4096, 2 = /2048,
                 // 3 = /1024, 4 = /512.
parameter        s12_wdlength = 11;    // Sectn 12 wordlength.
parameter        s11_wdlength = 12;    // Sectn 11 wordlength.
                 // s11 >= s12 >=wordlen


// --------------------------------------------------------------
//          Input/Output ports.
// --------------------------------------------------------------
```

```
        inc = 2'd0;
        //------------------------------------------------------------------
        endcase

5       always @(posedge clk)
        if (enable_3)
        begin
        if (!nrst || !rst)    // out_address=0 at end of line or pwr Reset.
          out_address_tmp <= 0;
10      else
          out_address_tmp <= out_address_tmp + inc;

        // Only count if at the correct point on line 2.
        if (in_address[rom_AddressSize-3] & (|in_address[rom_AddressSize-4:0]))
15        count <= ((count == 2'd2) ? 2'd0 : count + 2'd1); // Only count to 2.
        else
          count <= 2'd0;
        end

20      assign out_address = out_address_tmp;
        endmodule
```

<div align="center">Listing 12</div>

```
// SccsId: %W% %G%
25 /**************************************************************************
        Copyright (c) 1997 Pioneer Digital Design Centre Limited

   Author   : Dawood Alam.

30 Description: Verilog code for a configurable 2K/8K radix 2^2 + 2,
        singlepath-delay-feedback, decimation in frequency,
        (r22+2sdf DIF) Fast Fourier Transform (FFT) processor. (RTL)

   Notes   : This FFT processor computes one pair of I/Q data points every 4
35      fast clk cycles. A synchronous active-low reset flushes the
        entire pipeline and resets the FFT. Therefore the next pair of
        valid inputs are assumed to be the start of the active interval
        of the next symbol. There is a latency of 2048/8192 sample
        points + 7 slow clock cycles. This equates to (2048/8192 + 7)*4
40      fast clk cycles. When the out_ovf flag is raised an overflow has
        occured and saturation is performed on the intermediate
        calculation upon which the overflow has occured. If the valid_in
        flag is held low, the entire pipeline is halted and the
        valid_out flag is also held low. valid_out is also held low
45      until the entire pipeline is full (after the above number of
        clock cycles).

        To Do: RAM control (MUX),
            ROM lookup (quadrant lookup),
50              Change BF code for unique saturation nets for synthesis.
            ovf_detection (correct) register o/p
                ovf detection (correct) for mpy and BFs
                ROM/RAM test stuff.

55 **************************************************************************/
```

```
            enable_3;

        output [rom_AddressSize-2:0] out_address;

5       wire [rom_AddressSize-2:0] out_address;
        wire [1:0]      line_number;
        wire            nrst;

        reg [rom_AddressSize-2:0] out_address_tmp;
10      reg [1:0]       inc, count;
        reg             rst;

        // Decode which of the 4 lines are being addressed and assign it a line no.
        // Only need upper two bits of in_address since 4 lines in sequence length.
15      assign line_number = {in_address[rom_AddressSize-1],
                in_address[rom_AddressSize-2]};

        // Check for end of line and force out_address to zero on next clock edge.
        always @(in_address)
20      if (in_address[rom_AddressSize-3:0] == {rom_AddressSize-2{1'b1}})
          rst = 0;
        else
          rst = 1;

25      // Check for line number and decode appropriate out_address using algorithm
        // derived by studying coefficient tables for mpys M0, M1 and M2.
        always @(line_number or in_address or count)
        case (line_number)
        //-------------------------------------------------------------
30      2'd0: // LINE 0, inc by 2, then run the inc sequence 1,1,2,1,1,2...
          begin
            if (in_address[rom_AddressSize-3] & (|in_address[rom_AddressSize-4:0]))
            begin
              if (count == 2'd1 | count == 2'd0)
35            inc = 2'd1;
              else
              inc = 2'd2;
            end
            else
40          inc = 2'd2;
          end
        //-------------------------------------------------------------
        2'd1: // LINE 1, inc by 1.
          inc = 1;
45      //-------------------------------------------------------------
        2'd2: // LINE 2 inc by 3, (inc by 2 at N/4+1), (inc by 1 at N/2-1).
          begin
            if (in_address[rom_AddressSize-3:0] >= break_point3)
            inc = 2'd1;              // Third stage, inc by 1.
50          else if (in_address[rom_AddressSize-3:0] >= break_point2)
            inc = 2'd2;             // Second stage, inc by 2.
            else
            inc = 2'd3;             // First stage, inc by 3.
          end
55      //-------------------------------------------------------------
        2'd3: // LINE 3, fixed at address 0.
```

```
7'd33:b_tmp1 =20'b1000001010_1110011100; //W60_128=-0.980785 -0.195090
7'd34:b_tmp1 =20'b1000000010_1111001110; //W62_128=-0.995185 -0.098017
7'd88:b_tmp1 =20'b1000000001_1111100111; //W63_128=-0.998795 -0.049068
7'd89:b_tmp1 =20'b1000000010_0000110010; //W66_128=-0.995185 +0.098017
7'd90:b_tmp1 =20'b1000001111_0001111100; //W69_128=-0.970031 +0.242980
7'd91:b_tmp1 =20'b1000100111_0011000100; //W72_128=-0.923880 +0.382683
7'd92:b_tmp1 =20'b1001001001_0100000111; //W75_128=-0.857729 +0.514103
7'd93:b_tmp1 =20'b1001110100_0101000101; //W78_128=-0.773010 +0.634393
7'd94:b_tmp1 =20'b1010101000_0101111011; //W81_128=-0.671559 +0.740951
7'd95:b_tmp1 =20'b1011100100_0110101010; //W84_128=-0.555570 +0.831470
7'd96:b_tmp1 =20'b1100100101_0111001111; //W87_128=-0.427555 +0.903989
7'd97:b_tmp1 =20'b1101101011_0111101010; //W90_128=-0.290285 +0.956940
7'd98:b_tmp1 =20'b1110110101_0111111010; //W93_128=-0.146730 +0.989177
default:b_tmp1 =20'b0111111111_0000000000; //W00_128=+1.000000 -0.000000
endcase

always @(posedge clk)
if (enable_3)
  b_tmp2 <= b_tmp1;

assign out_br = b_tmp2[c_wordlength*2-1:c_wordlength];
assign out_bi = b_tmp2[c_wordlength-1:0];

endmodule
```

Listing 11

```
// SccsId: %W% %G%
/******************************************************************************
      Copyright (c) 1997 Pioneer Digital Design Centre Limited

Author   : Dawood Alam.

Description: Verilog code for a lookup table decoder.

Notes   : Used to generate addresses for each coefficient, based on the
          in_Address. Addresses are dependent on one of 4 rows
          (see figures) and on the sequence length (rom_AddressSize). Each
          row gives rise to a unique address sequence based on an
          algorithm. N refers to the index of the twiddle factor, NOT the
           absolute address. Breakpoints determine where inc values change
          on line 2.


******************************************************************************/

`timescale 1ns / 100ps

module fft_coeff_dcd (clk, enable_3, in_address, out_address, nrst);

parameter       rom_AddressSize = 1;   // Twice ROM address.
parameter       break_point2 = 1;      // 2nd break pt line 2
parameter       break_point3 = 1;      // 3rd break pt line 2

input [rom_AddressSize-1:0] in_address;
input       clk,
            nrst,
```

```
7'd49:b_tmp1 =20'b0110001100_1010111011; //W14_128=+0.773010 -0.634393
7'd50,
7'd72:b_tmp1 =20'b0101111011_1010101000; //W15_128=+0.740951 -0.671559
7'd11,
7'd51:b_tmp1 =20'b0101101010_1010010110; //W16_128=+0.707107 -0.707107
7'd52:b_tmp1 =20'b0101011000_1010000101; //W17_128=+0.671559 -0.740951
7'd12,
7'd73,
7'd53:b_tmp1 =20'b0101000101_1001110100; //W18_128=+0.634393 -0.773010
7'd54:b_tmp1 =20'b0100110001_1001100101; //W19_128=+0.595699 -0.803208
7'd13,
7'd55:b_tmp1 =20'b0100011100_1001010110; //W20_128=+0.555570 -0.831470
7'd74,
7'd56:b_tmp1 =20'b0100000111_1001001001; //W21_128=+0.514103 -0.857729
7'd14,
7'd57:b_tmp1 =20'b0011110001_1000111100; //W22_128=+0.471397 -0.881921
7'd58:b_tmp1 =20'b0011011011_1000110001; //W23_128=+0.427555 -0.903989
7'd15,
7'd75,
7'd59:b_tmp1 =20'b0011000100_1000100111; //W24_128=+0.382683 -0.923880
7'd60:b_tmp1 =20'b0010101100_1000011110; //W25_128=+0.336890 -0.941544
7'd16,
7'd61:b_tmp1 =20'b0010010101_1000010110; //W26_128=+0.290285 -0.956940
7'd76,
7'd62:b_tmp1 =20'b0001111100_1000001111; //W27_128=+0.242980 -0.970031
7'd17,
7'd63:b_tmp1 =20'b0001100100_1000001010; //W28_128=+0.195090 -0.980785
7'd64:b_tmp1 =20'b0001001011_1000000110; //W29_128=+0.146730 -0.989177
7'd18,
7'd77,
7'd65:b_tmp1 =20'b0000110010_1000000010; //W30_128=+0.098017 -0.995185
7'd66:b_tmp1 =20'b0000011001_1000000001; //W31_128=+0.049068 -0.998795
7'd19:b_tmp1 =20'b0000000000_1000000000; //W32_128=+0.000000 -1.000000
7'd78:b_tmp1 =20'b1111100111_1000000001; //W33_128=-0.049068 -0.998795
7'd20:b_tmp1 =20'b1111001110_1000000010; //W34_128=-0.098017 -0.995185
7'd79,
7'd21:b_tmp1 =20'b1110011100_1000001010; //W36_128=-0.195090 -0.980785
7'd22:b_tmp1 =20'b1101101011_1000010110; //W38_128=-0.290285 -0.956940
7'd80:b_tmp1 =20'b1101010100_1000011110; //W39_128=-0.336890 -0.941544
7'd23:b_tmp1 =20'b1100111100_1000100111; //W40_128=-0.382683 -0.923880
7'd81,
7'd24:b_tmp1 =20'b1100001111_1000111100; //W42_128=-0.471397 -0.881921
7'd25:b_tmp1 =20'b1011100100_1001010110; //W44_128=-0.555570 -0.831470
7'd82:b_tmp1 =20'b1011001111_1001100101; //W45_128=-0.595699 -0.803208
7'd26:b_tmp1 =20'b1010111011_1001110100; //W46_128=-0.634393 -0.773010
7'd83,
7'd27:b_tmp1 =20'b1010010110_1010010110; //W48_128=-0.707107 -0.707107
7'd28:b_tmp1 =20'b1001110100_1010111011; //W50_128=-0.773010 -0.634393
7'd84:b_tmp1 =20'b1001100101_1011001111; //W51_128=-0.803208 -0.595699
7'd29:b_tmp1 =20'b1001010110_1011100100; //W52_128=-0.831470 -0.555570
7'd85,
7'd30:b_tmp1 =20'b1000111100_1100001111; //W54_128=-0.881921 -0.471397
7'd31:b_tmp1 =20'b1000100111_1100111100; //W56_128=-0.923880 -0.382683
7'd86:b_tmp1 =20'b1000011110_1101010100; //W57_128=-0.941544 -0.336890
7'd32:b_tmp1 =20'b1000010110_1101101011; //W58_128=-0.956940 -0.290285
7'd87,
```

94

Description: Verilog code for 128 hardwired coefficients in a lookup table,
of which 64 are unique values.

5   Notes   : Used to store complex Twiddle factors. 128 point FFT twiddle
                factor coefficients (Radix 4+2). Coefficients stored as
                non-fractional 10 bit integers. Real Coefficient (cosine value)
                is coefficient high-byte. Imaginary Coefficient (sine value) is
                coefficient low-byte. Coefficient addresses are delayed by a
10   pipeline depth of 3, i.e. equivalent to case table values being
                advanced by 3.

        ***/

15   `timescale 1ns / 100ps

        module fft_hardwired_lu2 (clk, enable_3, address, out_br, out_bi);

        parameter   c_wordlength = 10;   // Coeff wordlength.
20   parameter   rom_AddressSize = 7;   // Address bus size.

        input   clk,
                        enable_3;
        input [rom_AddressSize-1:0] address;
25
        output [c_wordlength-1:0] out_br, out_bi;

        reg [c_wordlength*2-1:0] b_tmp1,
                b_tmp2;
30
        always @(address)
        case (address)
          7'd36:b_tmp1 =20'b0111111111_1111100111; //W01_128=+0.998795 -0.049068
          7'd4,
35   7'd37:b_tmp1 =20'b0111111110_1111001110; //W02_128=+0.995185 -0.098017
          7'd38,
          7'd68:b_tmp1 =20'b0111111010_1110110101; //W03_128=+0.989177 -0.146730
          7'd5,
          7'd39:b_tmp1 =20'b0111110110_1110011100; //W04_128=+0.980785 -0.195090
40   7'd40:b_tmp1 =20'b0111110001_1110000100; //W05_128=+0.970031 -0.242980
          7'd6,
          7'd41,
          7'd69:b_tmp1 =20'b0111101010_1101101011; //W06_128=+0.956940 -0.290285
          7'd42:b_tmp1 =20'b0111100010_1101010100; //W07_128=+0.941544 -0.336890
45   7'd7,
          7'd43:b_tmp1 =20'b0111011001_1100111100; //W08_128=+0.923880 -0.382683
          7'd44,
          7'd70:b_tmp1 =20'b0111001111_1100100101; //W09_128=+0.903989 -0.427555
          7'd8,
50   7'd45:b_tmp1 =20'b0111000100_1100001111; //W10_128=+0.881921 -0.471397
          7'd46:b_tmp1 =20'b0110110111_1011111001; //W11_128=+0.857729 -0.514103
          7'd9,
          7'd47,
          7'd71:b_tmp1 =20'b0110101010_1011100100; //W12_128=+0.831470 -0.555570
55   7'd48:b_tmp1 =20'b0110011011_1011001111; //W13_128=+0.803208 -0.595699
          7'd10,

```
       parameter        c_wordlength = 10;   // Coeff wordlength.
       parameter        rom_AddressSize = 5;  // Address bus size.

 5     input          clk,
                 enable_3;
       input [rom_AddressSize-1:0] address;

       output [c_wordlength-1:0]  out_br, out_bi;
10
       reg  [c_wordlength*2-1:0] b_tmp1,
                 b_tmp2;

       always @(address)
15     case (address)
         5'd5,
         5'd14:b_tmp1 = 20'b0111011001_1100111100;// W02_32 = +0.923880 -0.382683
         5'd6,
         5'd16:b_tmp1 = 20'b0101101010_1010010110;// W04_32 = +0.707107 -0.707107
20       5'd7,
         5'd18,
         5'd22:b_tmp1 = 20'b0011000100_1000100111;// W06_32 = +0.382683 -0.923880
         5'd8: b_tmp1 = 20'b0000000000_1000000000;// W08_32 = +0.000000 -1.000000
         5'd9: b_tmp1 = 20'b1100111100_1000100111;// W10_32 = -0.382683 -0.923880
25       5'd10,
         5'd24:b_tmp1 = 20'b1010010110_1010010110;// W12_32 = -0.707107 -0.707107
         5'd11:b_tmp1 = 20'b1000100111_1100111100;// W14_32 = -0.923880 -0.382683
         5'd13:b_tmp1 = 20'b0111110110_1110011100;// W01_32 = +0.980785 -0.195090
         5'd15,
30       5'd21:b_tmp1 = 20'b0110101010_1011100100;// W03_32 = +0.831470 -0.555570
         5'd17:b_tmp1 = 20'b0100011100_1001010110;// W05_32 = +0.555570 -0.831470
         5'd19:b_tmp1 = 20'b0001100100_1000001010;// W07_32 = +0.195090 -0.980785
         5'd23:b_tmp1 = 20'b1110011100_1000001010;// W09_32 = -0.195090 -0.980785
         5'd25:b_tmp1 = 20'b1000001010_1110011100;// W15_32 = -0.980785 -0.195090
35       5'd26:b_tmp1 = 20'b1000100111_0011000100;// W18_32 = -0.923880 +0.382683
         5'd27:b_tmp1 = 20'b1011100100_0110101010;// W21_32 = -0.555570 +0.831470
         default: b_tmp1 = 20'b0111111111_0000000000;// W00_32 = +1.000000 -0.000000
       endcase

40     always @(posedge clk)
        if (enable_3)
         b_tmp2 <= b_tmp1;

       assign out_br = b_tmp2[c_wordlength*2-1:c_wordlength];
45     assign out_bi = b_tmp2[c_wordlength-1:0];

       endmodule


50                                    Listing 10

       // SccsId: %W% %G%
       /*****************************************************************************
           Copyright (c) 1997 Pioneer Digital Design Centre Limited
55
       Author  : Dawood Alam.
```

```verilog
module fft_hardwired_lu0 (clk, enable_3, address, out_br, out_bi);

    parameter        c_wordlength = 10;   // Coeff wordlength.
    parameter        rom_AddressSize = 3;   // Address bus size.

    input       clk,
                enable_3;
    input [rom_AddressSize-1:0] address;

    output [c_wordlength-1:0]  out_br, out_bi;

    reg  [c_wordlength*2-1:0] b_tmp1,
             b_tmp2;

    always @(address)
    case (address)
     3'd6: b_tmp1 = 20'b0000000000_1000000000; // W2_8 = +0.000000 -1.000000
     3'd0: b_tmp1 = 20'b0101101010_1010010110; // W1_8 = +0.707107 -0.707107
     3'd2: b_tmp1 = 20'b1010010110_1010010110; // W3_8 = -0.707107 -0.707107
     default:b_tmp1 = 20'b0111111111_0000000000;// W0_8 = +1.000000 -0.000000
    endcase

    always @(posedge clk)
     if (enable_3)
      b_tmp2 <= b_tmp1;

    assign out_br = b_tmp2[c_wordlength*2-1:c_wordlength];
    assign out_bi = b_tmp2[c_wordlength-1:0];

endmodule
```

Listing 9

```verilog
// Sccsld: %W% %G%
/*****************************************************************************
      Copyright (c) 1997 Pioneer Digital Design Centre Limited

   Author   : Dawood Alam.

   Description: Verilog code for 32 hardwired coefficients in a lookup table, of
        which 16 are unique values.

   Notes   : Used to store complex Twiddle factors. 32 point FFT twiddle
        factor coefficients (Radix 4+2). Coefficients stored as
        non-fractional 10 bit integers. Real Coefficient (cosine value)
        is coefficient high-byte. Imaginary Coefficient (sine value) is
        coefficient low-byte. Coefficient addresses are delayed by a
        pipeline depth of 4, i.e. equivalent to case table values being
        advanced by 4.


   *****************************************************************************/

`timescale 1ns / 100ps

module fft_hardwired_lu1 (clk, enable_3, address, out_br, out_bi);
```

```verilog
output [wordlength-1:0]  out_xr,          // SR output data I.
            out_xi;          // SR output data Q.

reg   [wordlength-1:0]  shift_r [length-1:0];   // SR for I data.
reg   [wordlength-1:0]  shift_i [length-1:0];   // SR for Q data/

wire  [wordlength-1:0]  out_xr,
            out_xi;
wire            clk,
            enable_3;
integer         i;

always @ (posedge clk)
 if (enable_3)
   begin
     for (i = (length-1); i >= 0; i = i - 1)
      begin
       if (i == 0)
        begin
       shift_r[0] <= in_xr;       // Force input I to SR.
            shift_i[0] <= in_xi;         // Force input Q to SR.
         end
         else
         begin
       shift_r[i] <= shift_r[i-1];    // Shift data I once.
            shift_i[i] <= shift_i[i-1];    // Shift data Q once.
         end
        end
      end
    end

assign out_xr = shift_r[length-1];
assign out_xi = shift_i[length-1];
endmodule
```

                                    Listing 8

```verilog
// SccsId: %W% %G%
/*******************************************************************************
       Copyright (c) 1997 Pioneer Digital Design Centre Limited

Author   : Dawood Alam.

Description: Verilog code for 8 hardwired coefficients in a lookup table, of
       which 4 are unique values.

Notes   : Used to store complex Twiddle factors. 8 point FFT twiddle factor
       coefficients (Radix 4+2). Coefficients stored as non-fractional
       10 bit integers. Real Coefficient (cosine value) is coefficient
       high-byte. Imaginary Coefficient (sine value) is coefficient
       low-byte. Coefficient addresses are delayed by a pipeline depth
       of 5, i.e. equivalent to case table values being advanced by 5.


*******************************************************************************/


`timescale 1ns / 100ps
```

```
// Register ovf outputs before final OR, else whole complex multiplier is
// treated as combinational, and the intermediate pipeline reg is ignored.
always @(posedge clk)
// if (enable_0 || enable_1 || enable_2)
   ovf_tmp2 <= ovf_tmp0;

always @(posedge clk)
 ovf_tmp3 <= ovf_tmp1;

assign out_ovf = ovf_tmp2 || ovf_tmp3;

`ifdef OVERFLOW_DEBUG_LOW_LEVEL
// Debug code to display overflow output of a particular adder.
// Concurrently monitor overflow flag and halt on overflow.
always @(posedge clk)
 if (out_ovf)
  begin
   if (ovf_tmp2) $display("ovf_tmp0 on complex multiplier = ",ovf_tmp2);
   if (ovf_tmp3) $display("ovf_tmp1 on complex multiplier = ",ovf_tmp3);
   $stop.
  end
`else
`endif

assign out_cr = cr_tmp;
assign out_ci = ci_tmp;

endmodule
```

Listing 7

```
// SccsId: %W% %G%
/********************************************************************

       Copyright (c) 1997 Pioneer Digital Design Centre Limited

  Author   : Dawood Alam.

  Description: Verilog code for a dual-port FIFO with complex data store. (RTL)

  Notes    : A variable bitwidth FIFO shift register for intermediate I/Q
             calculations.

********************************************************************/

`timescale 1ns / 100ps

module fft_sr_iq (clk, enable_3, in_xr, in_xi, out_xr, out_xi);

parameter       wordlength = 1;    // Data wordlength I/Q.
parameter       length = 1;      // Shift reg length.

input        clk,          // Master clock;
             enable_3;       // Enable on clock 3.
input [wordlength-1:0] in_xr,        // SR input data, I.
             in_xi;       // SR input data, Q.
```

```
      begin
        coeff_tmp1 = in_ar_tmp;
        sum_tmp0 = store_tmp;
      end
5   endcase

    abri_tmp3 = {{wordlength{abri_tmp2[wordlength-1]}},abri_tmp2}; // extnd
    coeff_tmp2 = {{wordlength{coeff_tmp1[wordlength-1]}},coeff_tmp1};// extnd
    mpy_tmp2 = (abri_tmp3 * coeff_tmp2);
10
    mpy_tmp1 = mpy_tmp2[wordlength*2-mult_scale:wordlength-(mult_scale-1)];

    if (c4)
      {ex_reg1,sum_tmp2} = sum_tmp0 - mpy_tmp1 - mpy_tmp2[wordlength-mult_scale];
15  else
      {ex_reg1,sum_tmp2}        =       mpy_tmp1       +       sum_tmp0       +
    mpy_tmp2[wordlength-mult_scale];

    ovf_tmp1 = (c4 ^ mpy_tmp1[wordlength-1]) &&      // Overflow check.
20        sum_tmp0[wordlength-1] &&           // Deals with a
          ~sum_tmp2[wordlength-1] ||          // +/- input.
          ~(c4 ^ mpy_tmp1[wordlength-1]) &&
          ~sum_tmp0[wordlength-1] &&
          sum_tmp2[wordlength-1];
25    if (ovf_tmp1)                // Saturate logic.
        sum_tmp1 = (ex_reg1) ? {1'b1,{wordlength-1{1'b0}}} :
            {1'b0,{wordlength-1{1'b1}}};
      else
        sum_tmp1 = sum_tmp2;
30  end

    // Pipeline registers for I/Q data paths and intermediate registers.
    always @(posedge clk)
      begin
35      if (enable_2) // Enable on 2nd clock.
        acc_tmp <= sum_tmp1;            // Temp store.

        if (enable_3) // Enable on 3rd clock.
        cr_tmp <= acc_tmp;              // Pipeline reg cr
40
        if (enable_3) // Enable on 3rd clock.
        ci_tmp <= sum_tmp1;            // Pipeline reg ci

        if(enable_1)
45      store_tmp <= sum_tmp1;          // Temp store.

        if (enable_2)
        in_ar_tmp <= in_ar;            // Reg i/p to mpy.

50      if (enable_1)
        in_ai_tmp <= in_ai;            // Reg i/p to mpy.

        if (enable_0 || enable_1 || enable_2)
        abri_tmp2 <= abri_tmp1;        // Pipeline reg.
55
      end
```